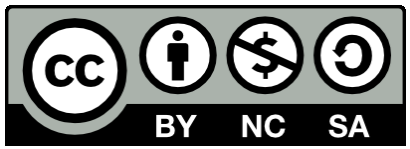
Gestion de code avec Git (FIEC30EM)
Utiliser Git avec GitHub

frank.buloup@univ-amu.fr

Aix Marseille Université
Faculté des Sciences du Sport
Master 2 IEAP - Facteurs Humains



amU
Aix Marseille Université

cnrs
advancing the frontiers

INSTITUT DES SCIENCES DU MOUVEMENT ETIENNE JULES MAREY

Gestion de code avec Git

1. Pour aller plus loin
   - Utilisation sur GitHub
   - Les zones de Git
   - Annuler un commit : commande revert
   - Modifier l'historique des commits : commande reset
   - Fusionner et rebaser des branches - Résolution de conflits
   - Collaboration

Exercice I - Premier pas avec GitHub

1. Créer un compte sur GitHub
2. Créer un premier dépôt public dans GitHub nommé "FirstRepository"

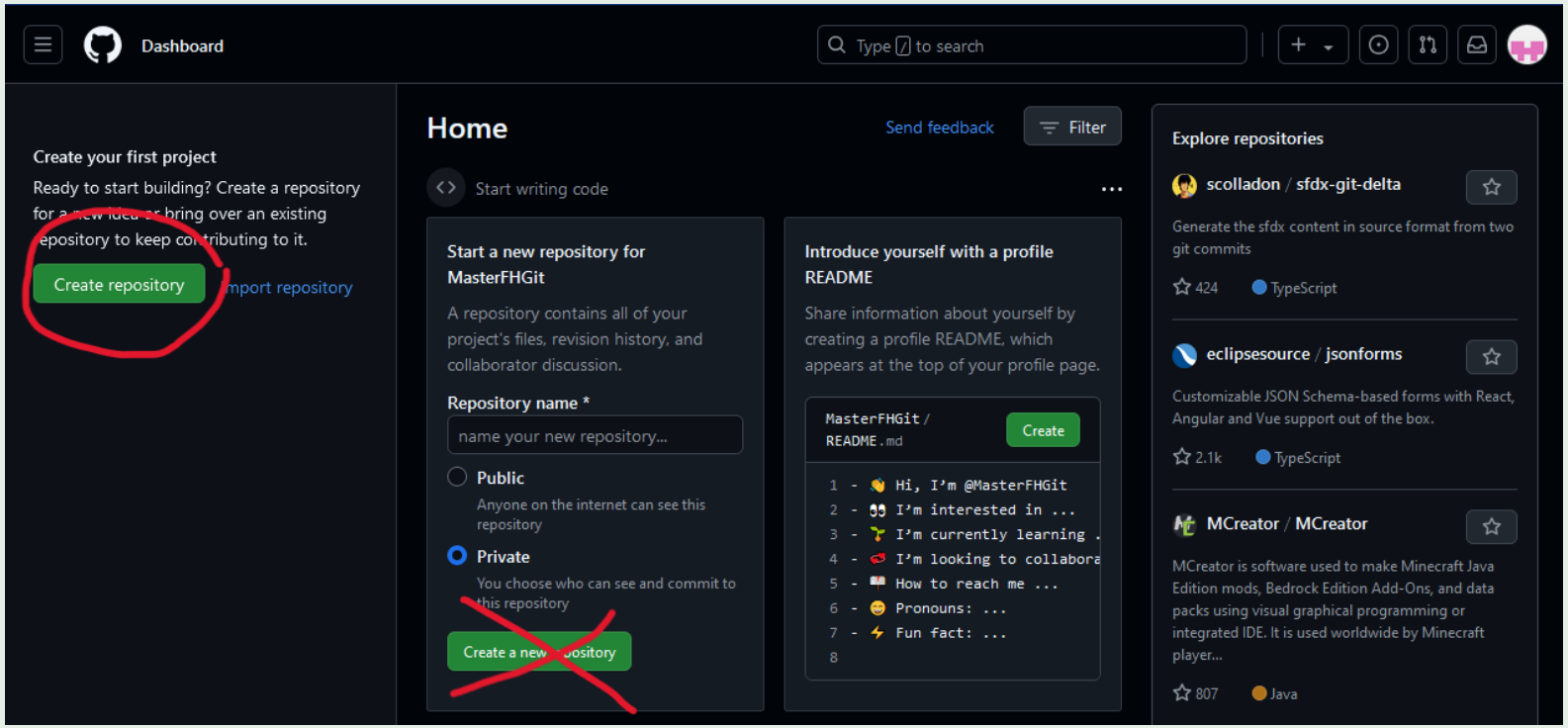


Figure: Création d'un dépôt dans Github

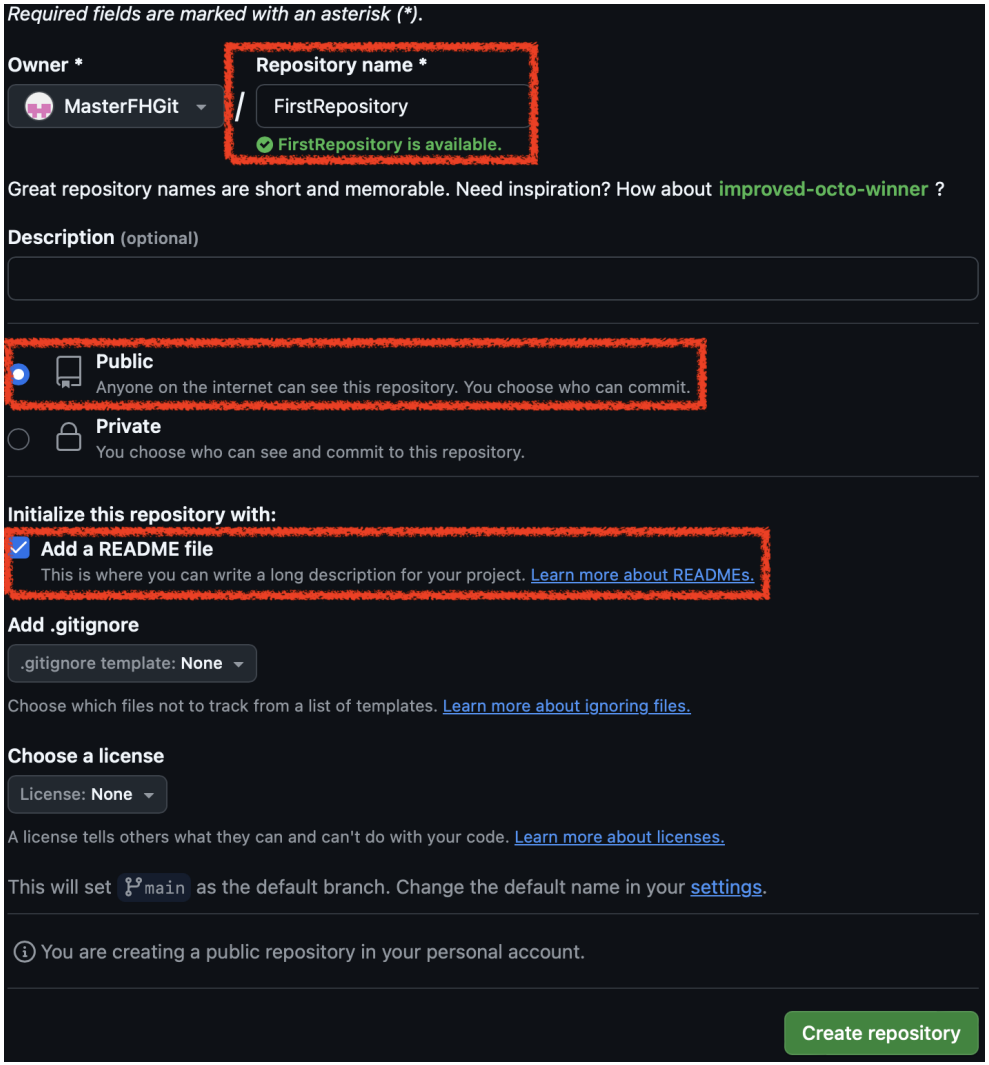
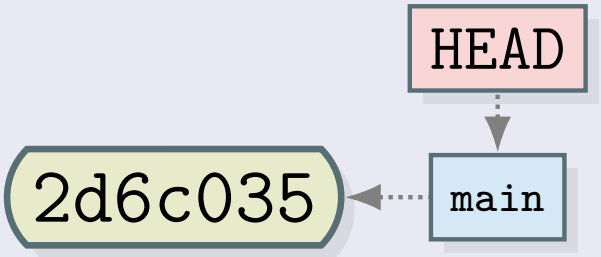


Figure: Création du dépôt "FirstRepository" public avec fichier README

État initial après le premier commit sur GitHub



- **2d6c035** est l'ID du commit (l'ID complet est un SHA1)
- **main** est une référence au dernier commit de la branche qui porte le même nom, sur le dépôt distant
- On peut avoir plusieurs branches et donc plusieurs références (**main**, **dev**, **feature1** etc.)
- On trouvera aussi **master** à la place de **main**
- **main** ou **master** sont par convention des noms de branches de production

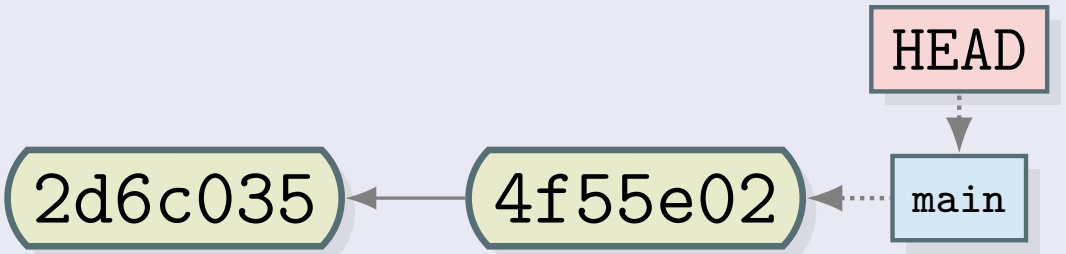
Exercice I - Premier pas avec GitHub

3. Modifier le fichier README.md sur GitHub

Exercice I - Premier pas avec GitHub

3. Modifier le fichier README.md sur GitHub

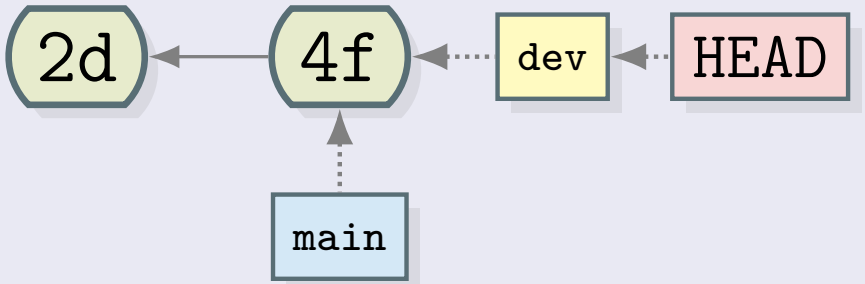
État après le deuxième commit sur github



Exercice I - Premier pas avec GitHub

4. Créer une nouvelle branche nommée **dev**
5. Sur la branche **dev**, modifier le fichier README.md

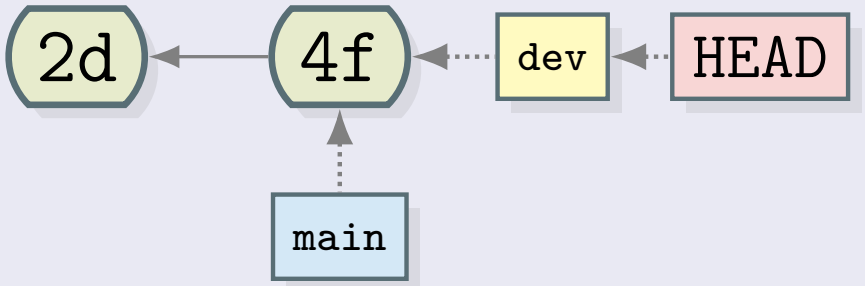
État juste après la création de la branche **dev**



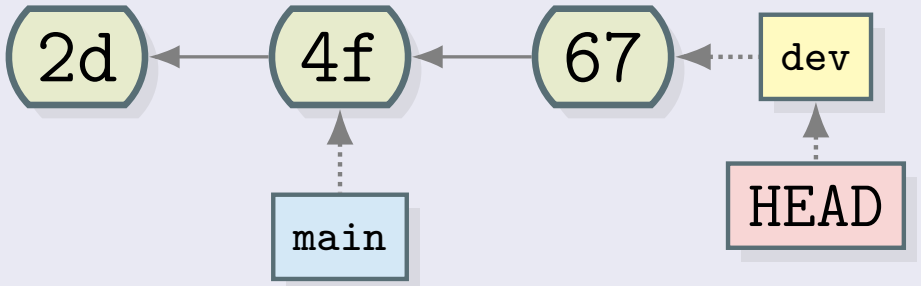
Exercice I - Premier pas avec GitHub

4. Créer une nouvelle branche nommée **dev**
5. Sur la branche **dev**, modifier le fichier README.md

État juste après la création de la branche **dev**



État juste après le commit dans la branche **dev**



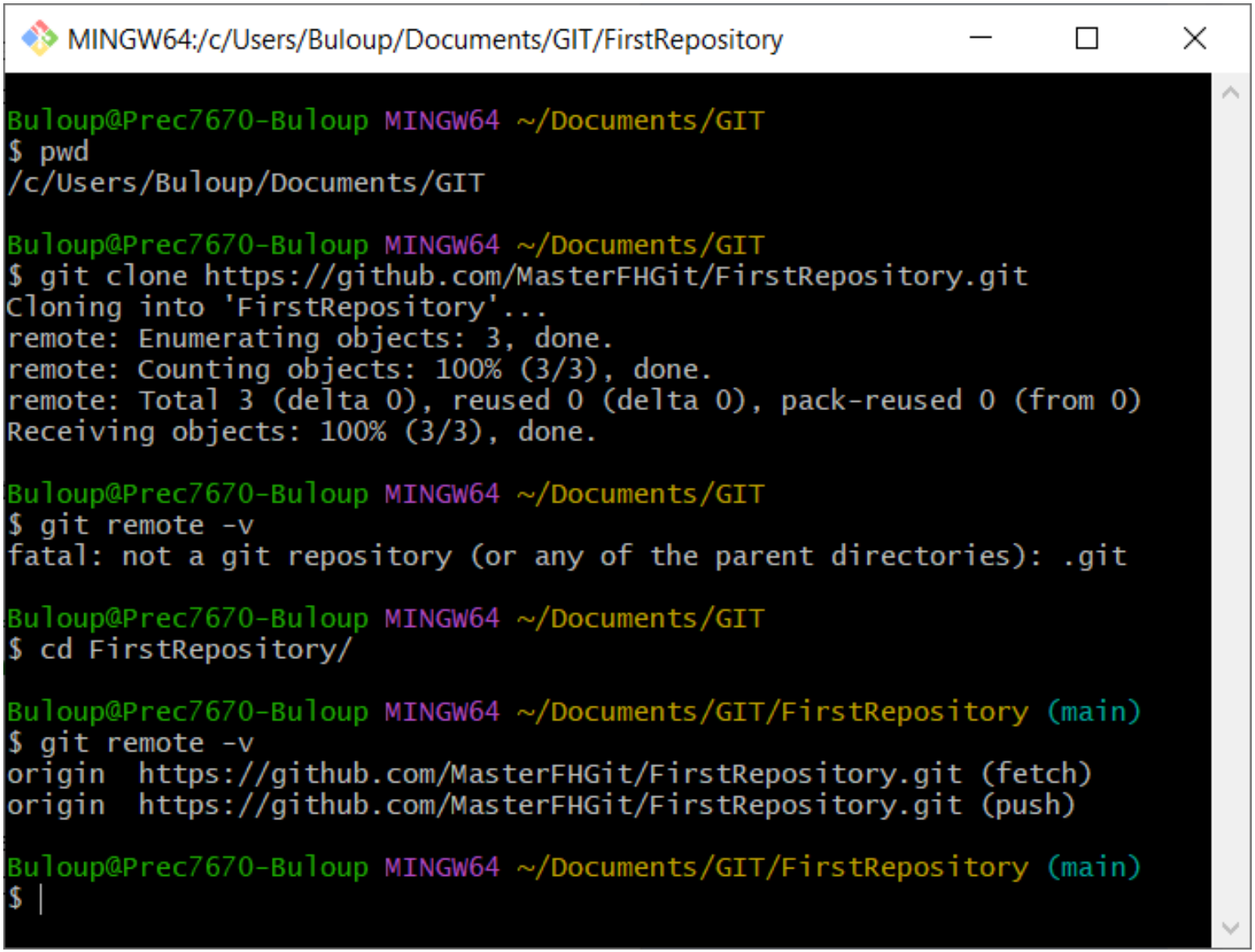


Figure: Commandes clone et remote

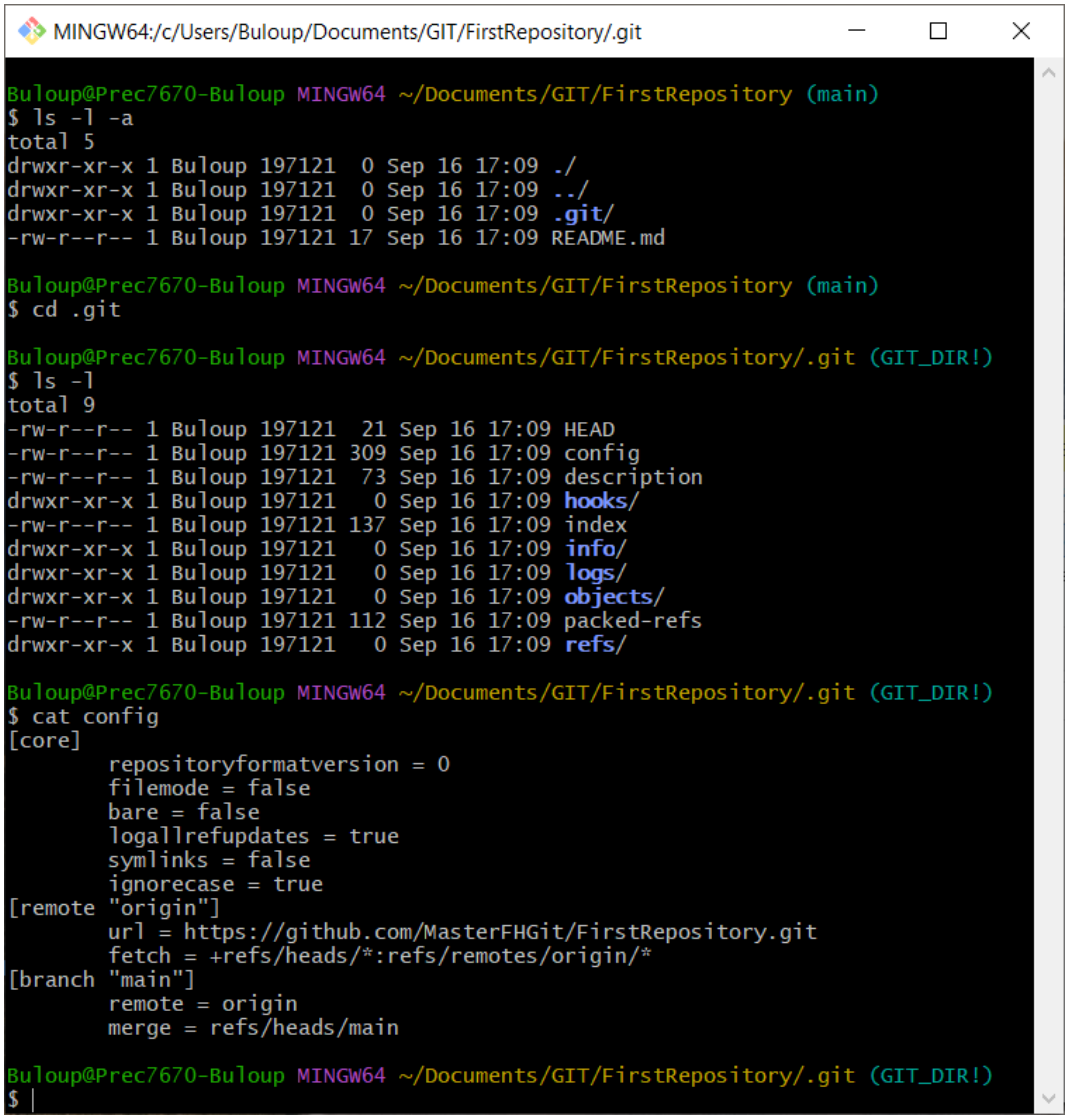


Figure: Dossier .git et fichier config

remote "origin" ?

- origin est un alias local donné au dépôt distant
- il peut être changé dans le fichier config ...

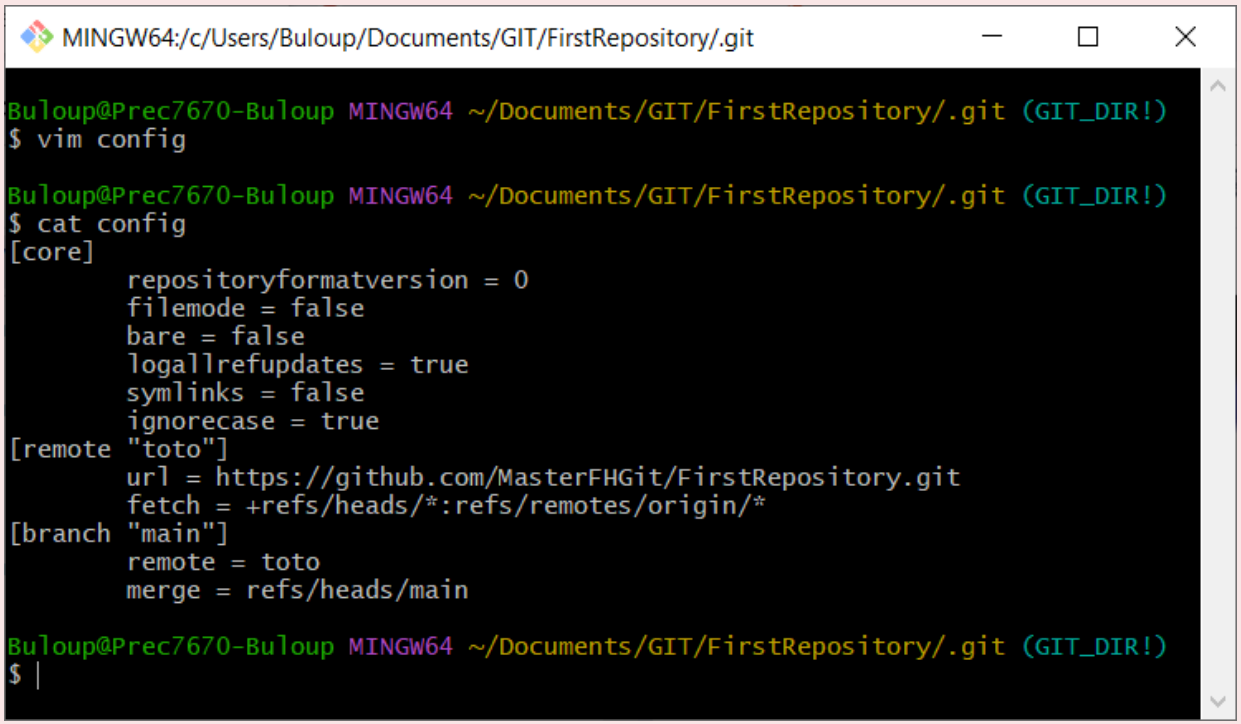


Figure: Changer l'alias origin

... ou lors de la commande clone

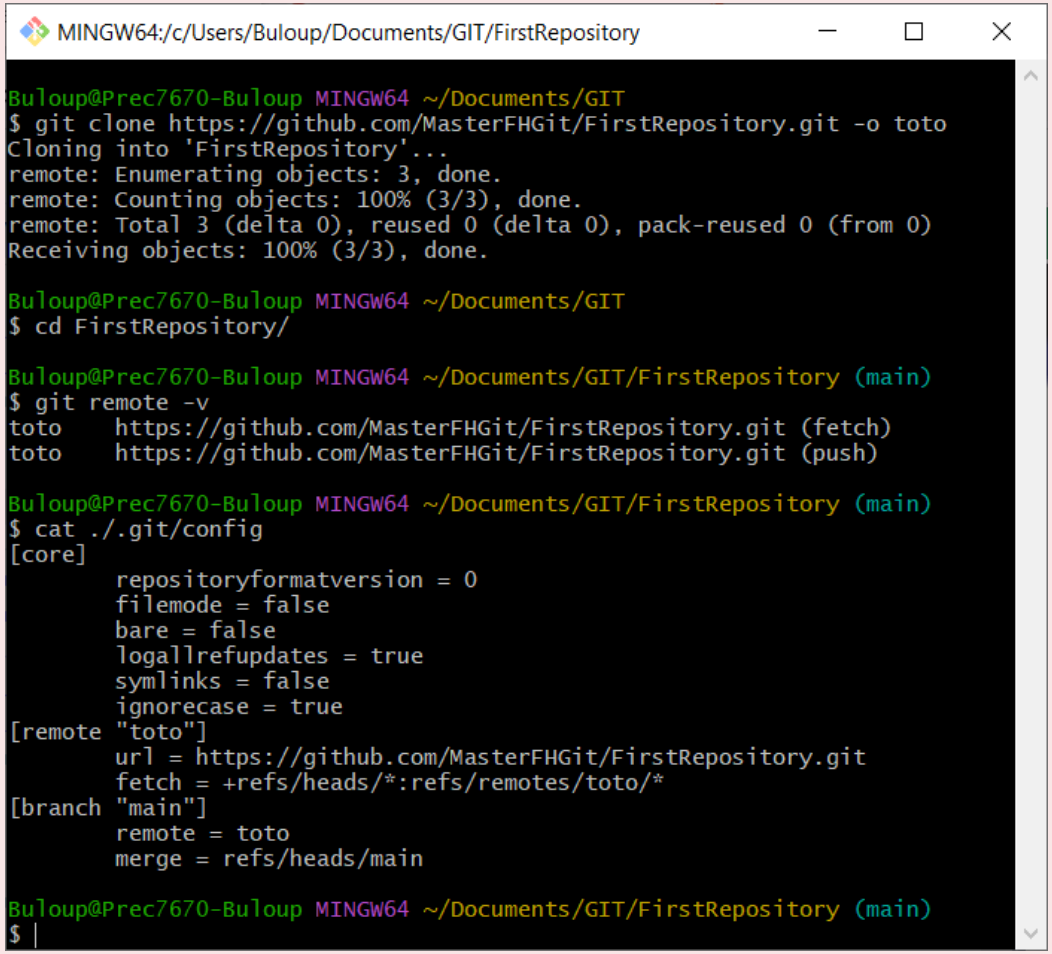


Figure: Changer l'alias origin

Comment supprimer un dépôt local ?

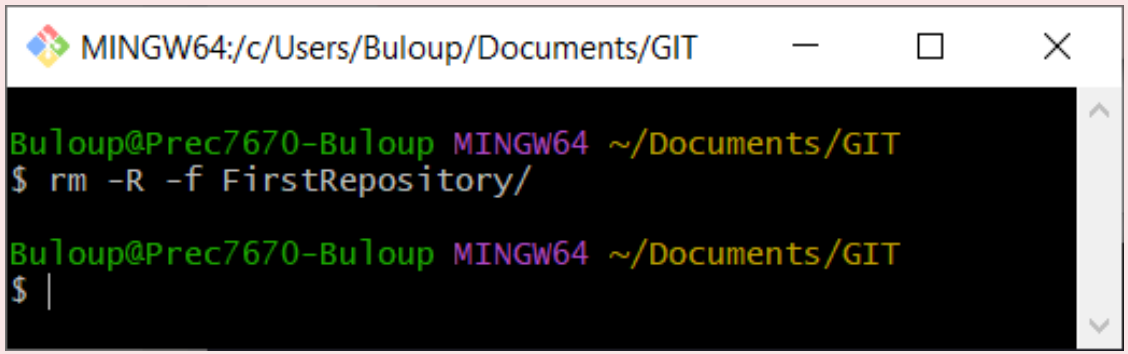


Figure: Supprimer un dépôt local

branch, main, fetch, push, merge etc. ?

C'est le vocabulaire des DVCS (Git)
Plus clair progressivement !

Petit résumé intermédiaire

Quelques Commandes : clone, remote

```
1 $ git clone url              # Clone repository
2 $ git remote -v              # Remotes infos
```

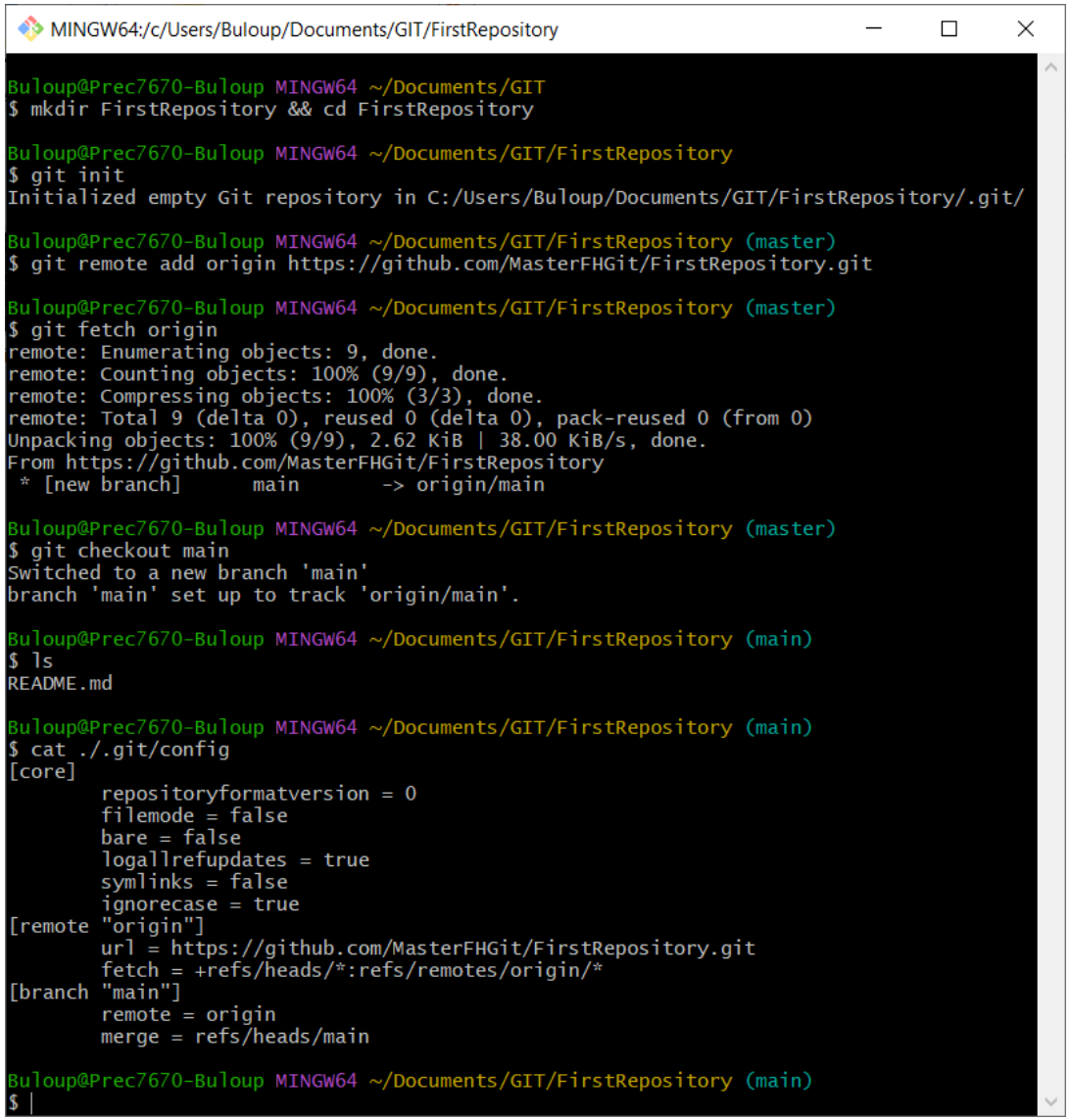


Figure: Importer un dépôt distant - Version 2

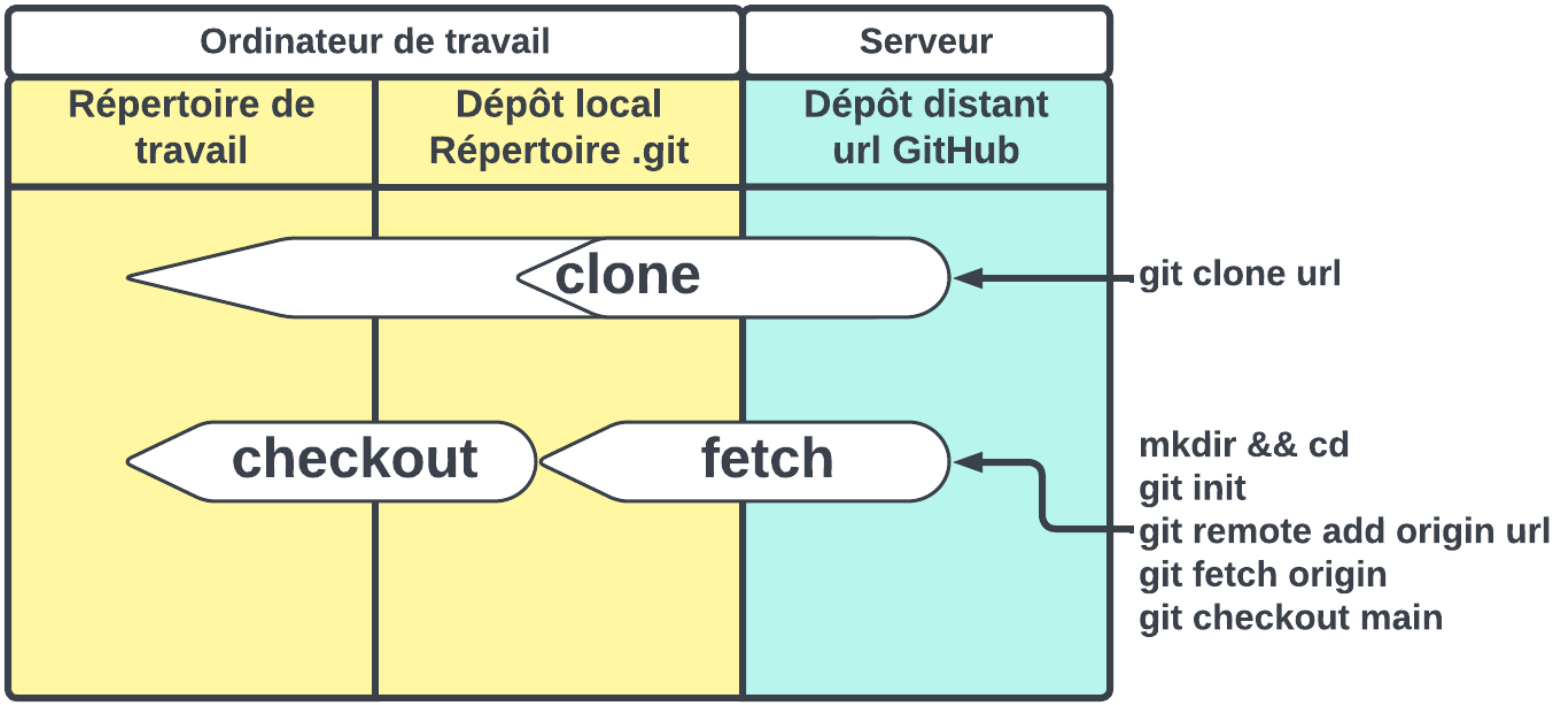


Figure: Importer un dépôt distant - Les deux versions

Exercice I - Changements à partir du dépôt distant

1. Changer le fichier README sur github (branche main)
2. Reporter ces changements en local

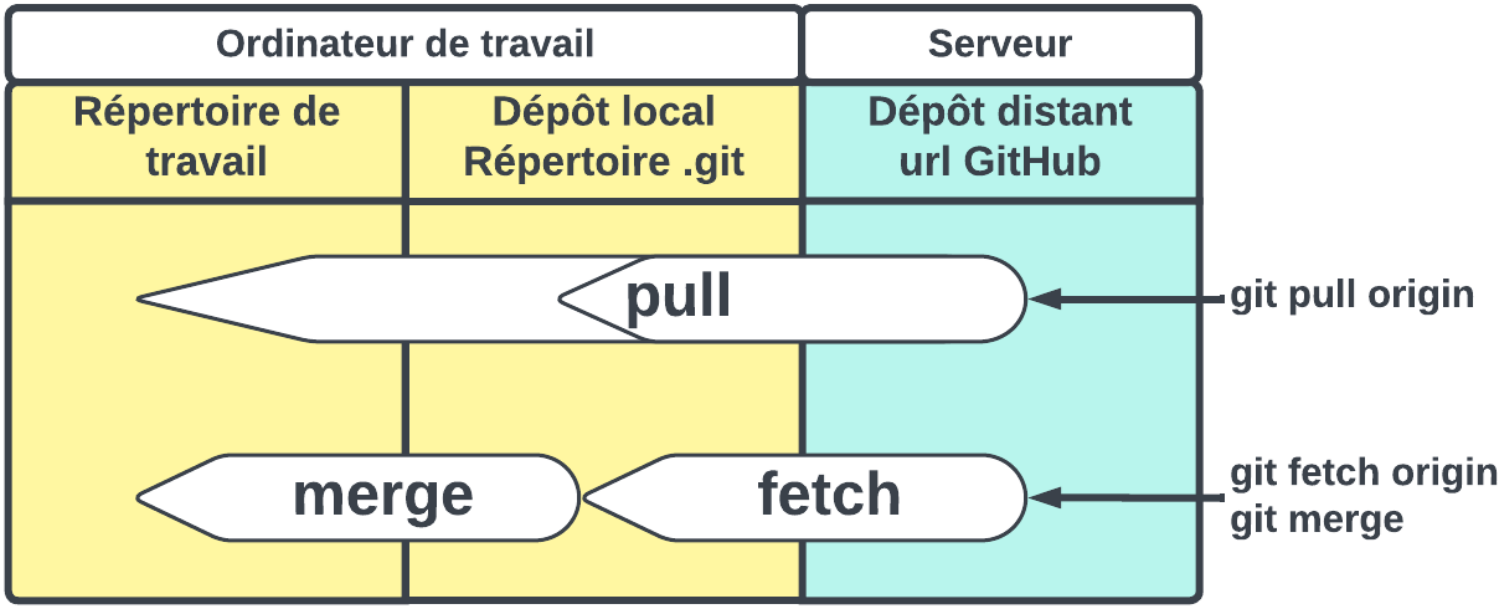


Figure: Mettre à jour un dépôt local - Les deux versions

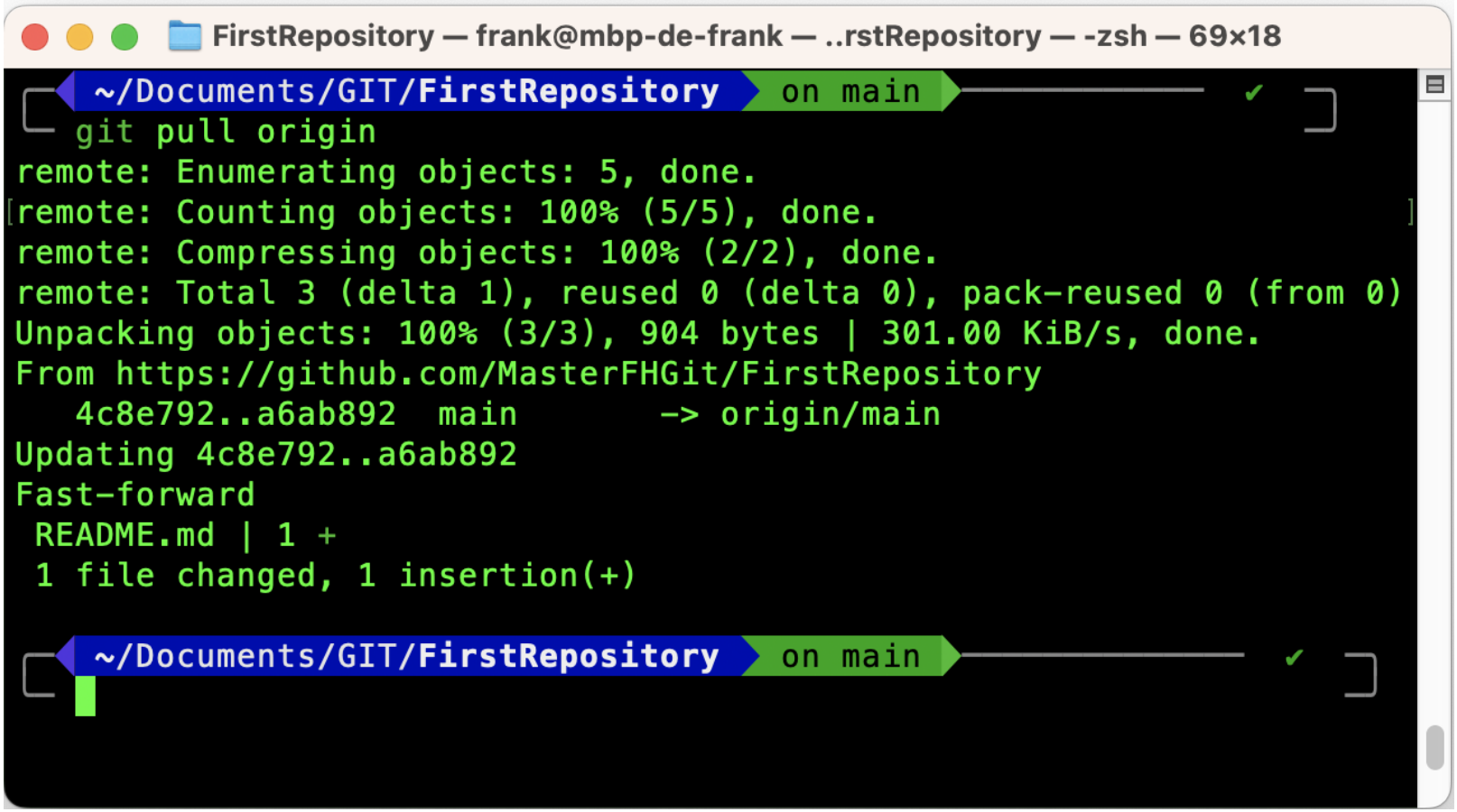


Figure: Mettre à jour un dépôt local - Version 1

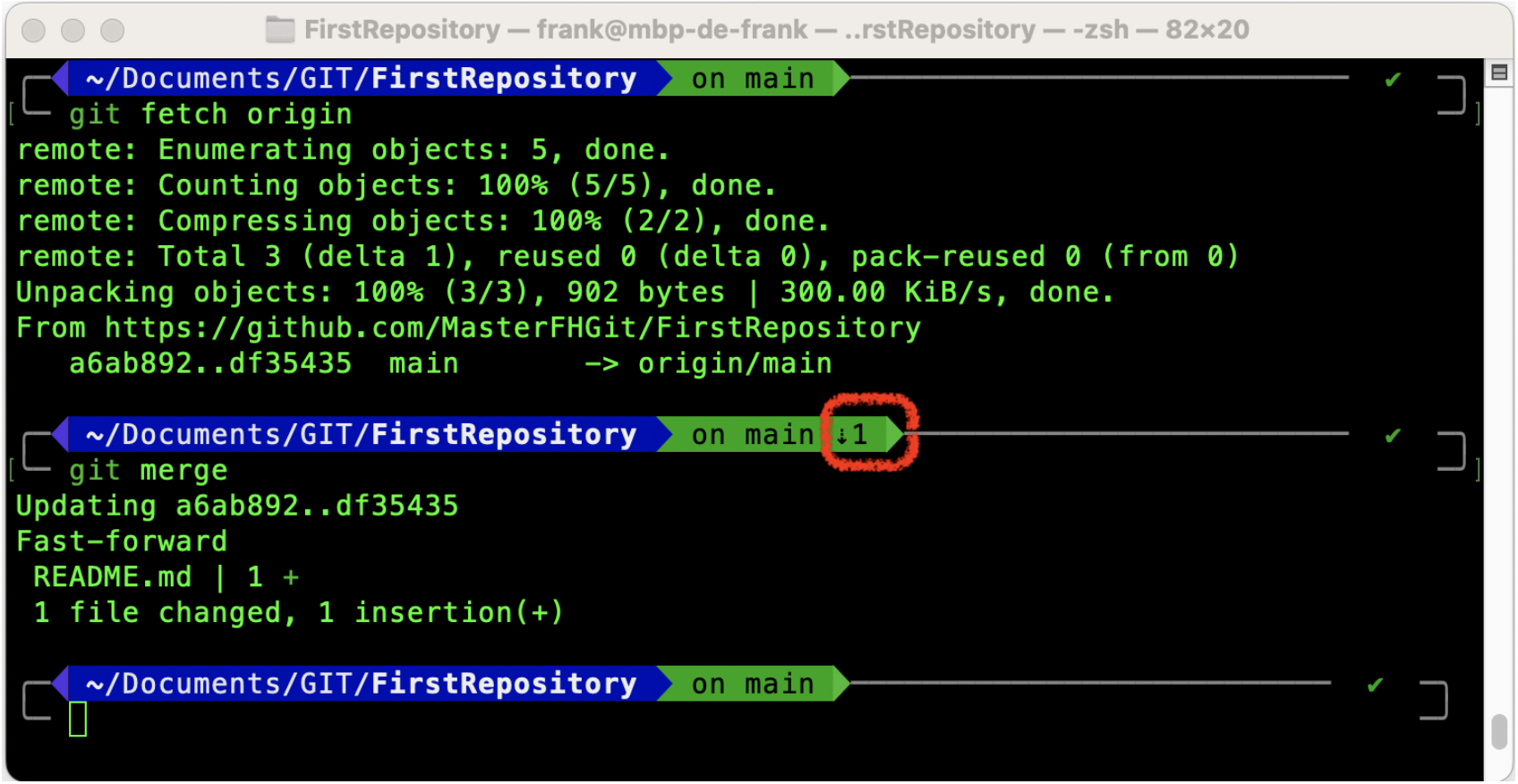


Figure: Mettre à jour un dépôt local - Version 2

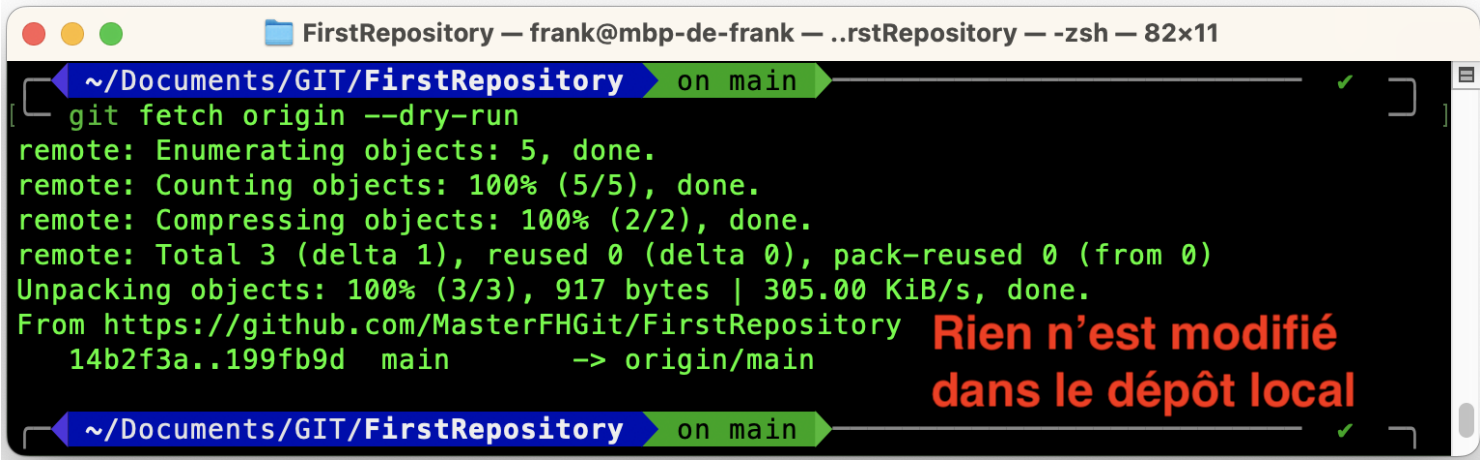


Figure: Le dépôt local est-il à jour ? Non !

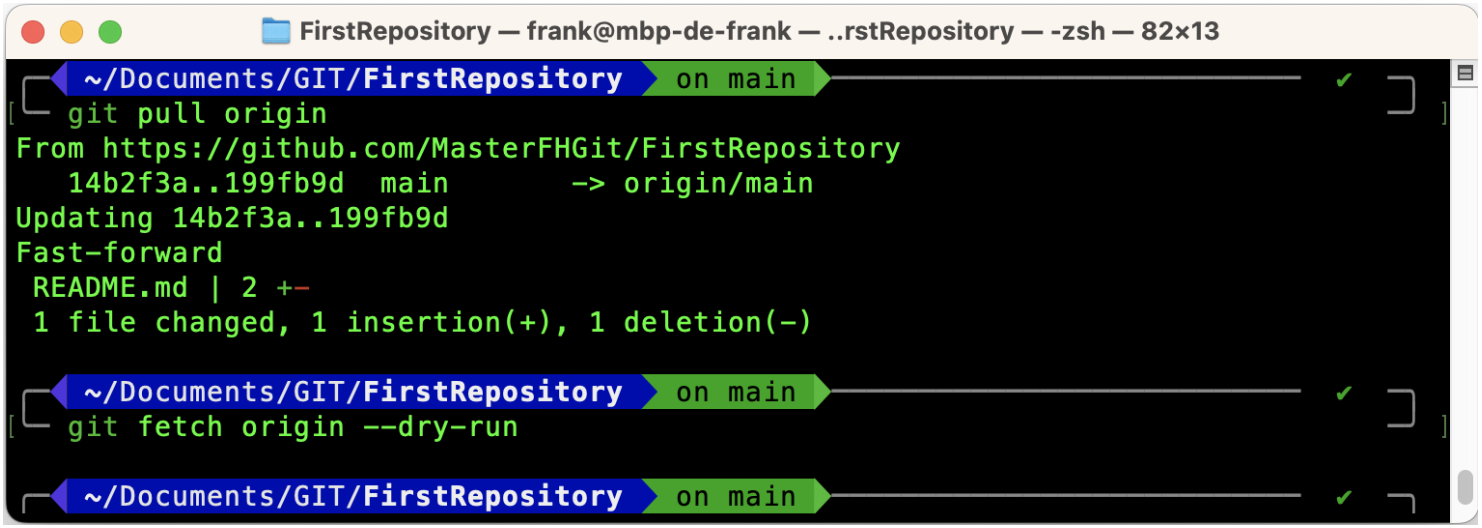


Figure: Maintenant oui !

Petit résumé intermédiaire

Quelques commandes : pull, fetch

```
$ git pull origin                    # Get from origin
$ git fetch origin && git merge      # Idem !
$ git fetch origin --dry-run         # Has origin changed ?
$ git log                            # Print commits
$ git log --oneline                  # Print abbrev-commits
```

Exercice II - Changements à partir du dépôt local

1. Changer le fichier README en local (branche main)
2. Reporter ces changements en distant

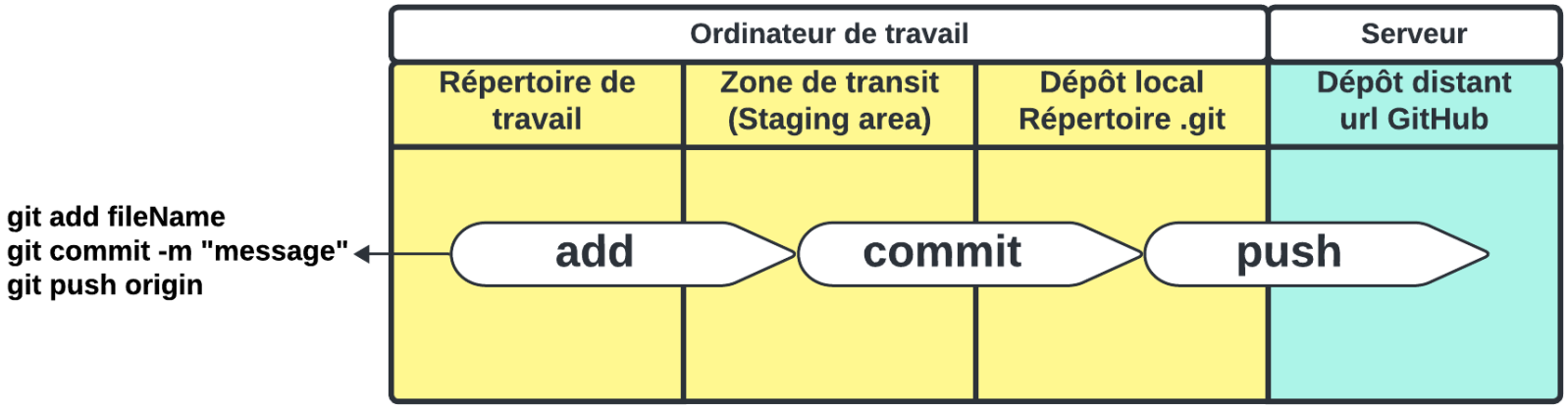


Figure: Mettre à jour le dépôt distant

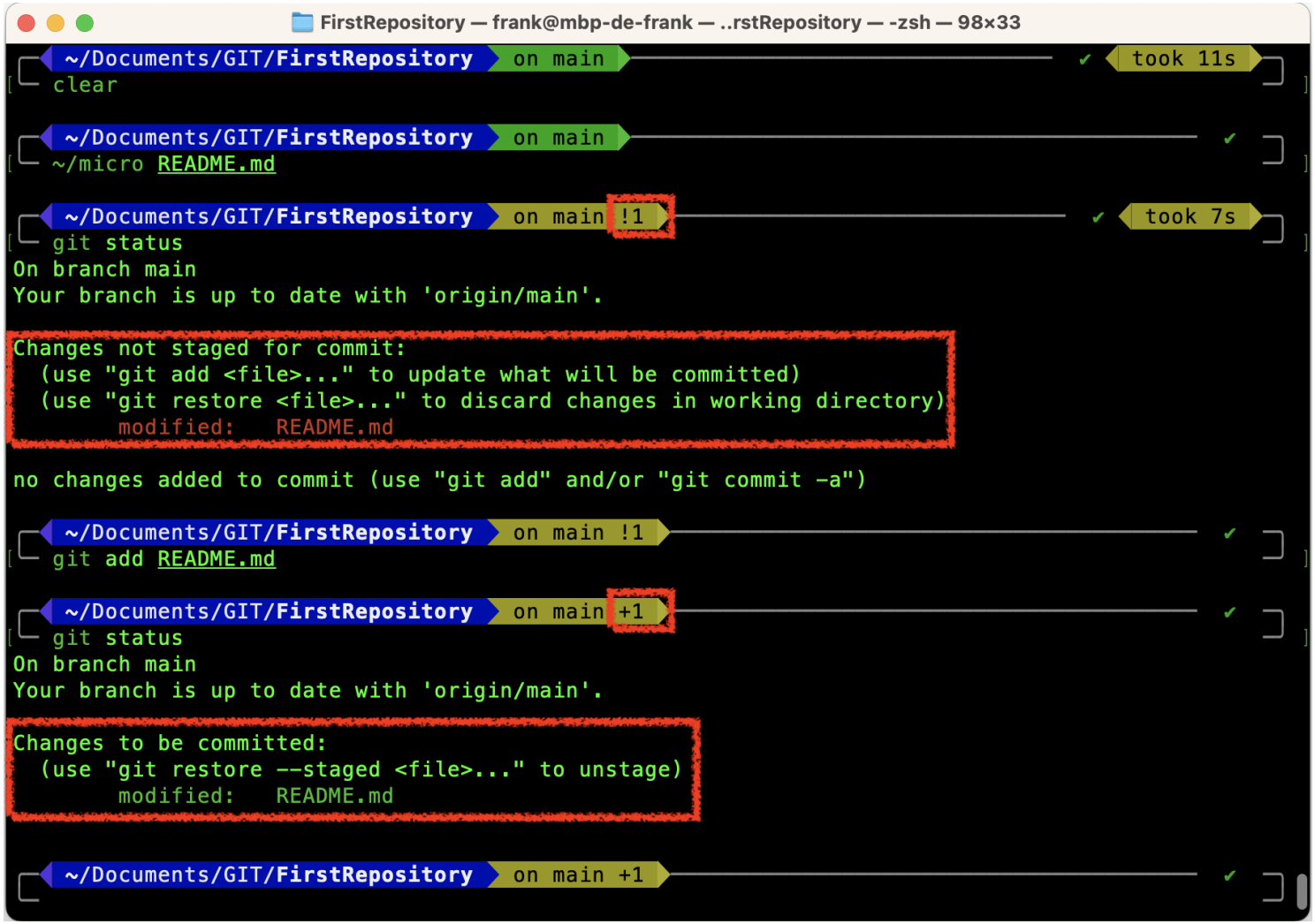


Figure: Zone de transit

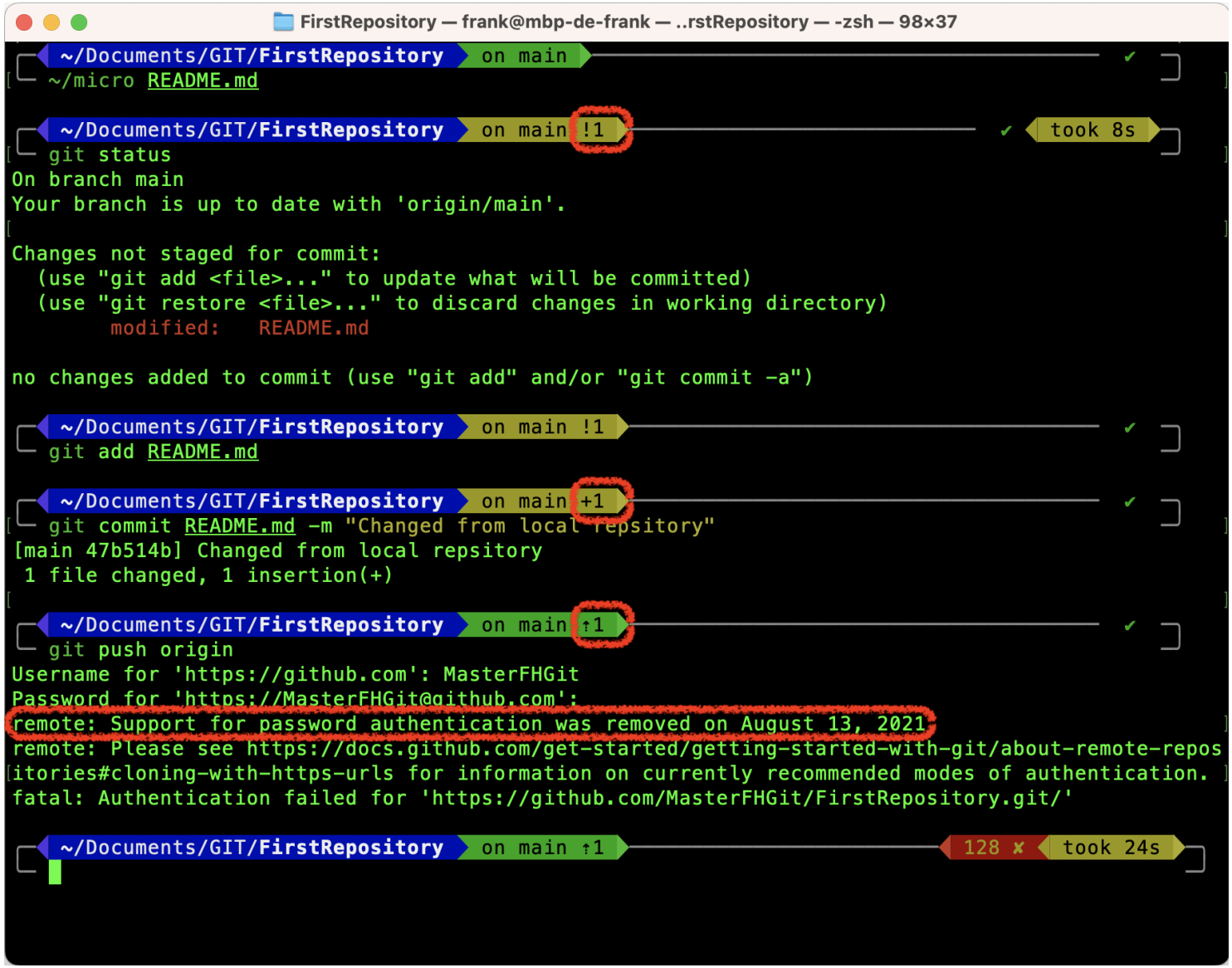


Figure: Mettre à jour le dépôt distant

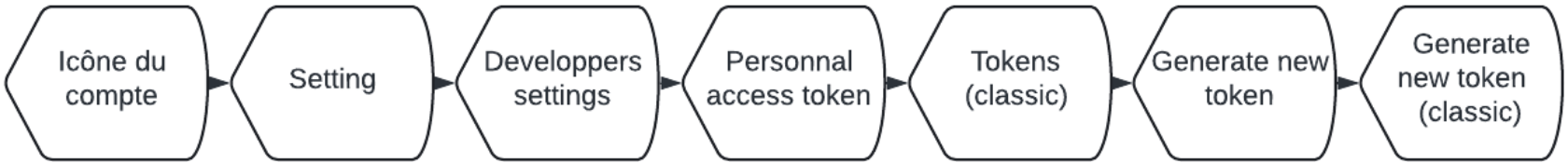


Figure: Génération d'un jeton dans GitHub

Génération d'un jeton d'accès personnel

- Conserver ce jeton. Il n'est visible qu'une seule fois !
- C'est bien de lui donner un petit nom (note) !
- Ne pas oublier de sélectionner l'option "repo"
- Sur un dépôt privé, rien n'est possible sans jeton

Pour plus d'infos : **Personal Access Token (PAT)**

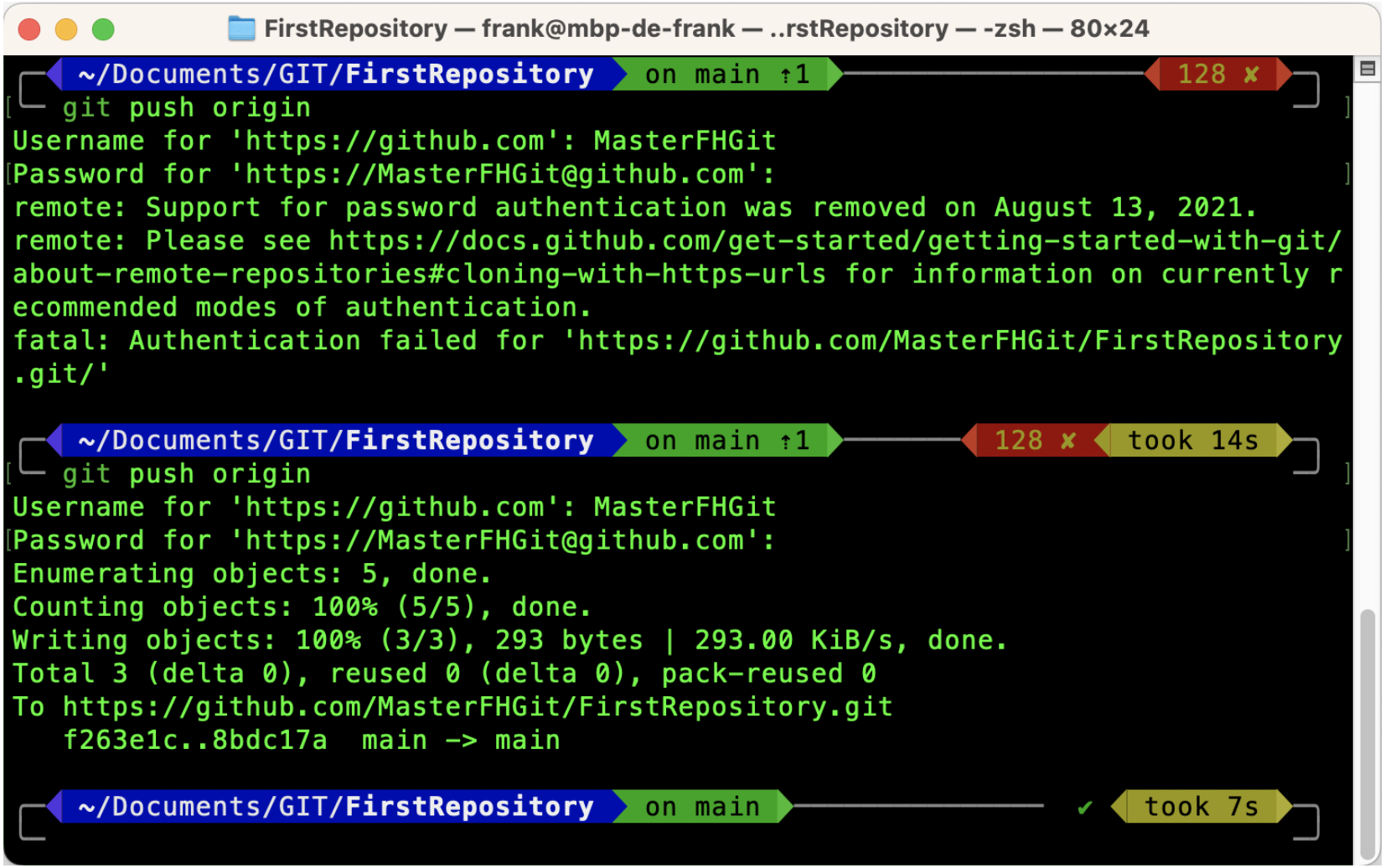


Figure: Avec le jeton comme mdp, le push fonctionne

Pour enregistrer le jeton

```
1 $ git config --global credential.helper store
```

... et ne plus avoir à le retaper qu'une seule fois. Le jeton est enregistré dans le fichier texte **.git-credentials** de votre répertoire personnel.

Pour signer les commits automatiquement

```
1 $ git config --global user.email "mail@somewhere.com"
2 $ git config --global user.name "username"
```

Signed-off-by: username <mail@somewhere.com>

Pour cloner (ou autre) on peut aussi faire :

```
1 $ git clone https://uname:pass@github.com/uname/rp.git
```

Petit résumé intermédiaire

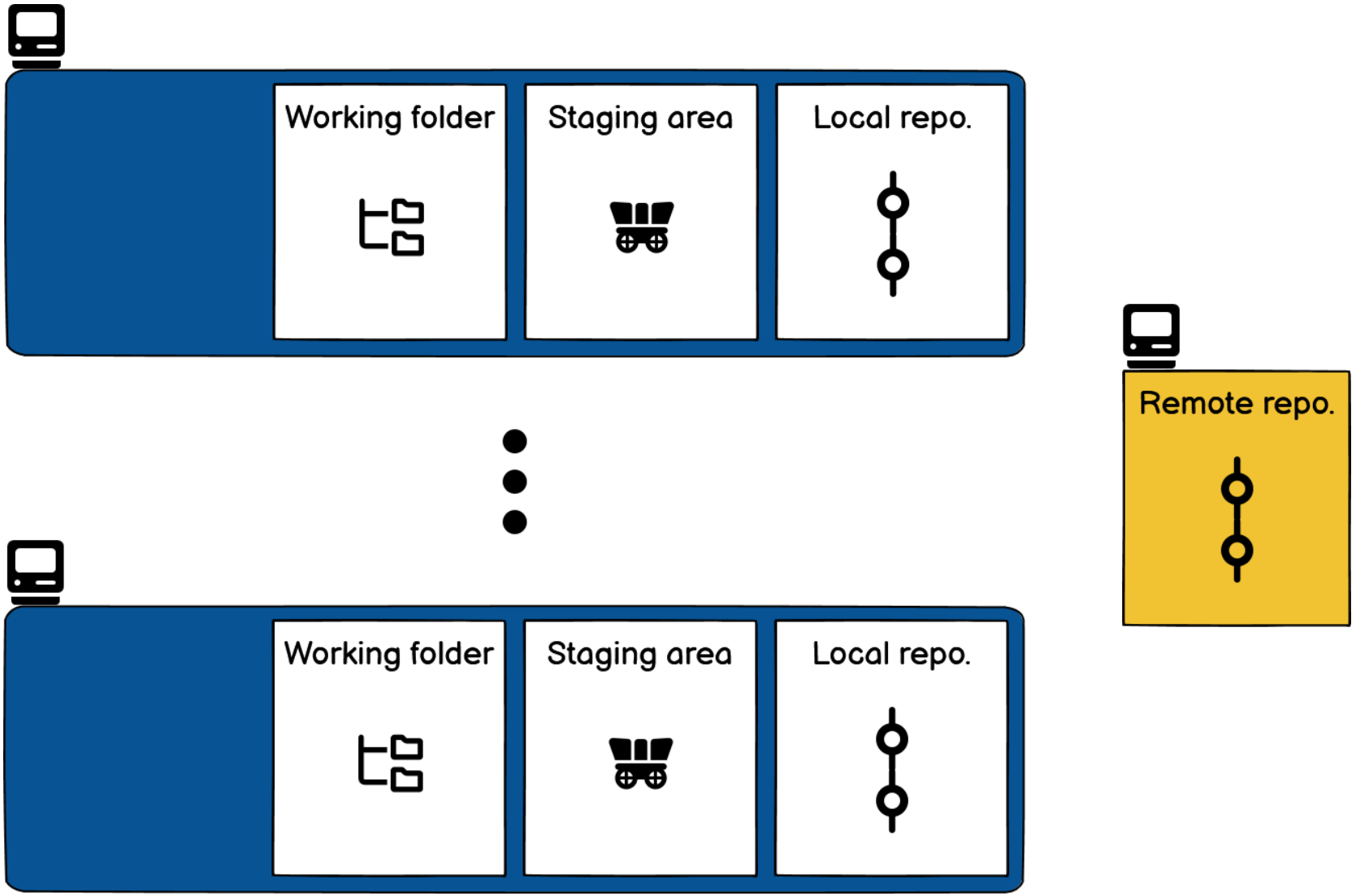
Quelques commandes : add, commit

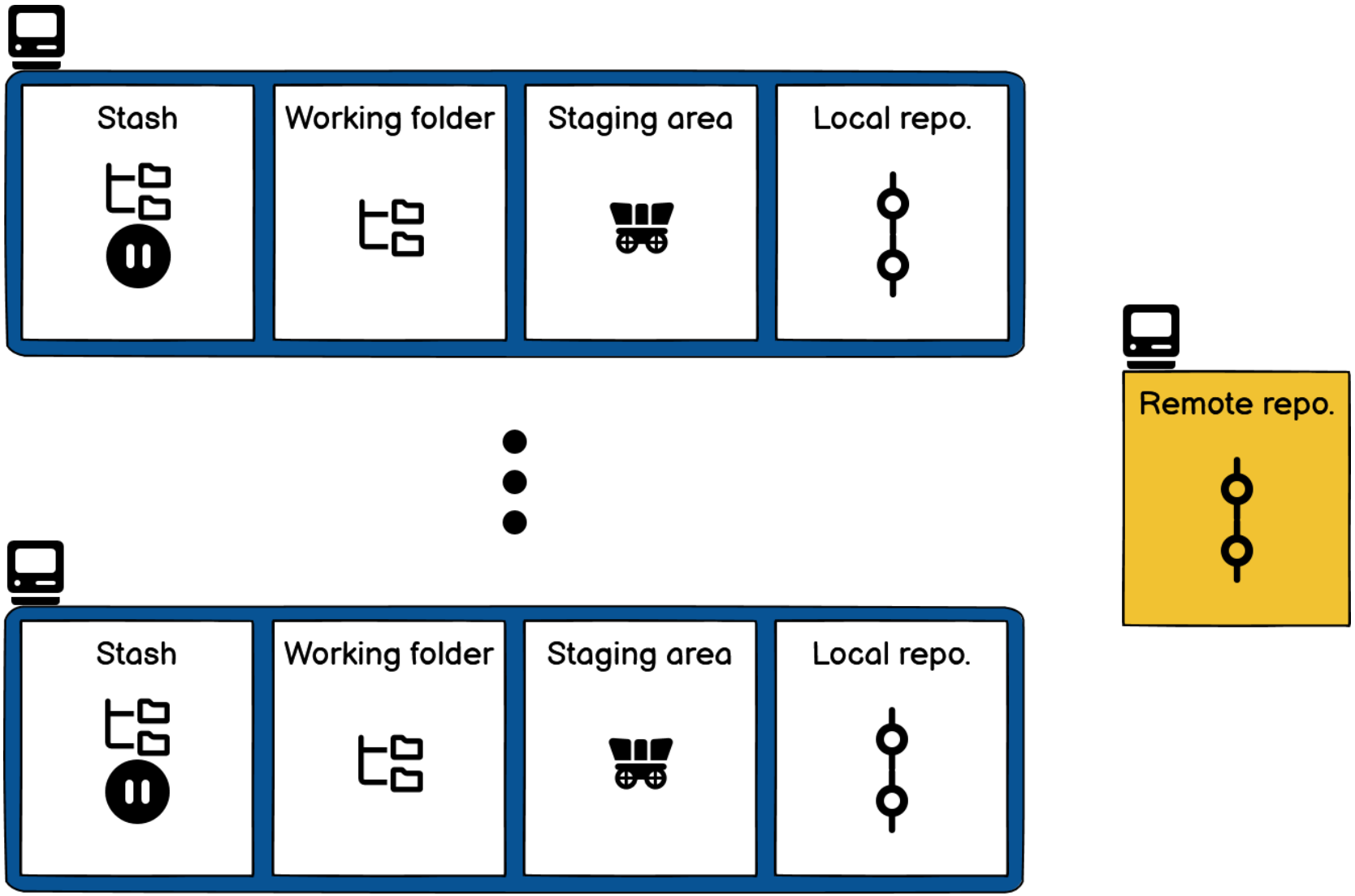
```
$ git add fileName # Stage fileName
$ git add .        # Stage modified and new files
$ git add -u       # Stage modified and deleted files
$ git add -A       # Stage everything
$ git commit -m "A commit message"
```

D'autres commandes utiles ! switch, branch, checkout

```
$ git switch branchName      # Switch to branchName
$ git checkout 52fd6e4       # In detached HEAD
$ git checkout branchName    # HEAD on branchName
$ git branch -D branchName   # To delete branchName
$ git branch                 # List all local branches
$ git branch -r              # List all remote branches
$ git branch -a              # List all branches
$ git branch newBranchName   # Create new branch
$ git switch -c neBrNa       # Create & switch to neBrNa
```

Les zones de Git





stash = réserve, remise

Exercice III - Commande revert

1. Ajouter un nouveau commit dans la branche **main** en modifiant le fichier README.md (Before revert)
2. Utiliser la commande revert pour annuler ce commit
3. Utiliser la commande checkout pour vous déplacer dans les commits

Exercice III - Commande revert

1. Ajouter un nouveau commit dans la branche **main** en modifiant le fichier README.md (Before revert)
2. Utiliser la commande revert pour annuler ce commit
3. Utiliser la commande checkout pour vous déplacer dans les commits

Commande revert

```
$ git switch main              # Goto main branch
$ vim README.md                # Modify README
$ git commit -am "Before revert" # Commit
$ git revert HEAD              # Revert last commit
$ git log --oneline            # See logs
$ cat README.md                # File content
$ git checkout 44e5761         # Goto previous commit
$ cat README.md                # File content
```

Commande revert

- Elle ne modifie pas l'historique des commits (pas de suppression de commits)
- Elle crée un nouveau commit
- Elle détermine comment inverser les changements introduits par le commit spécifié et ajoute un nouveau commit avec le contenu inversé qui en résulte

Exercice IV - Commande reset

1. Modifier le fichier README.md sans faire de commit
2. Supprimer les deux derniers commits (le Before revert et le revert) précédents en utilisant la commande reset
3. Conclusion

Exercice IV - Commande reset

1. Modifier le fichier README.md sans faire de commit
2. Supprimer les deux derniers commits (le Before revert et le revert) précédents en utilisant la commande reset
3. Conclusion

Commande reset

```
$ git checkout main     # HEAD to main
$ vim README.md         # Change README.md
$ git log --oneline     # See logs
$ git reset 9d49258     # Reset two commits
$ git log --oneline     # See logs
$ cat README.md         # File content
$ git restore README.md # Get from index
$ cat README.md         # File content
```

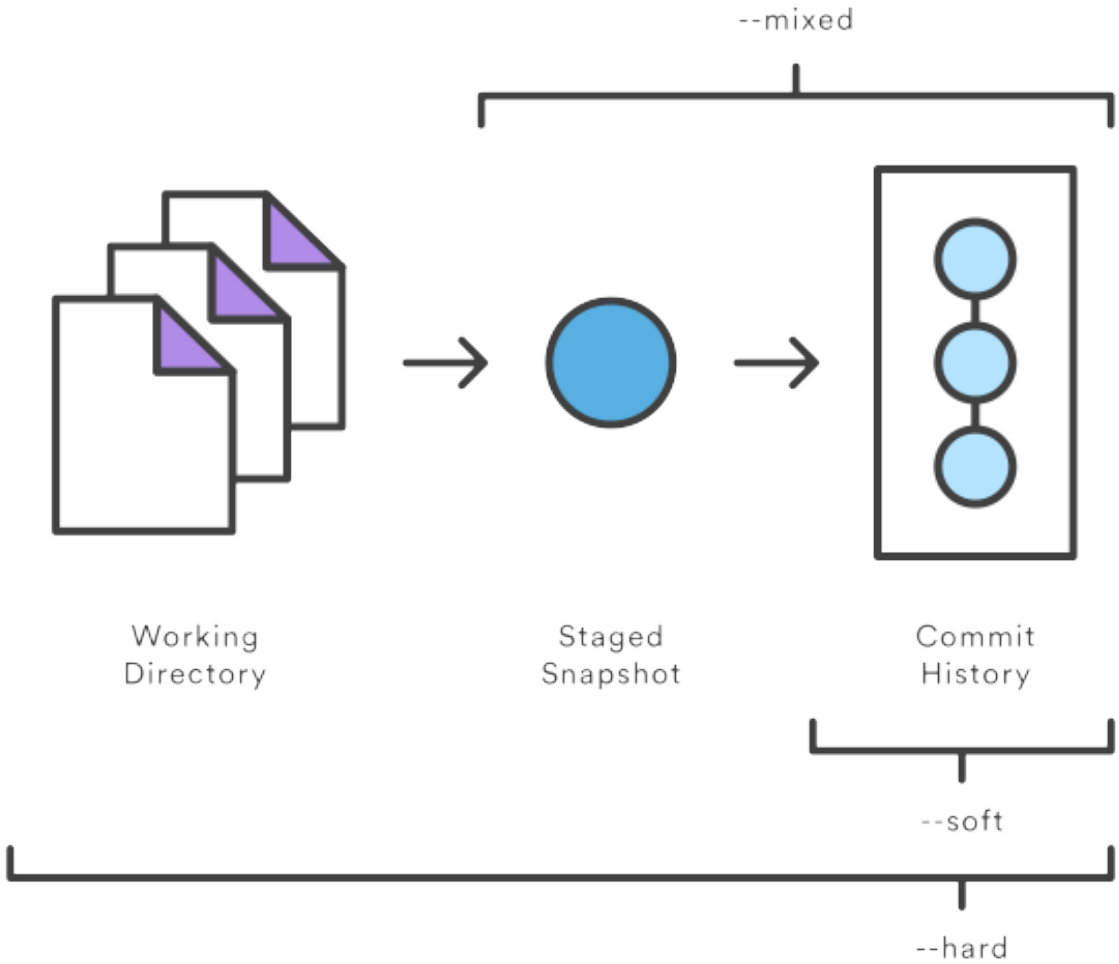


Figure: Commande reset - Les différents modes

Commande reset - Option mixed

- Elle revient dans l'état du commit spécifié en mettant l'index (staging area) à jour mais pas la copie de travail
- Elle modifie l'historique des commits (supprime des commits)
- La commande restore permet de récupérer l'index : git restore README.md
- C'est l'option par défaut : mode mixte

"git reset idCommit" et "git reset --mixed idCommit" sont donc des commandes indentiques

Commande reset - Options soft, mixed et hard

- Récupérer l'état de certains fichiers et conserver les modifications en cours pour les autres fichiers
- Réécrire l'historique des commits dans le dépôt (e.g. regrouper des commits)

Comment annuler un reset ?

Quelle que soit l'option utilisée, il est toujours possible d'annuler un reset en retrouvant l'ID des commits par la commande reflog

```
1 $ git reflog  # get idCommit just before reset
2 $ # reset hard to this commit
3 $ git reset --hard idCommit
```

--hard reste une option dangereuse !

Exercice V - Préparation fusion et rebase

1. Ajouter un nouveau commit dans la branche **dev**

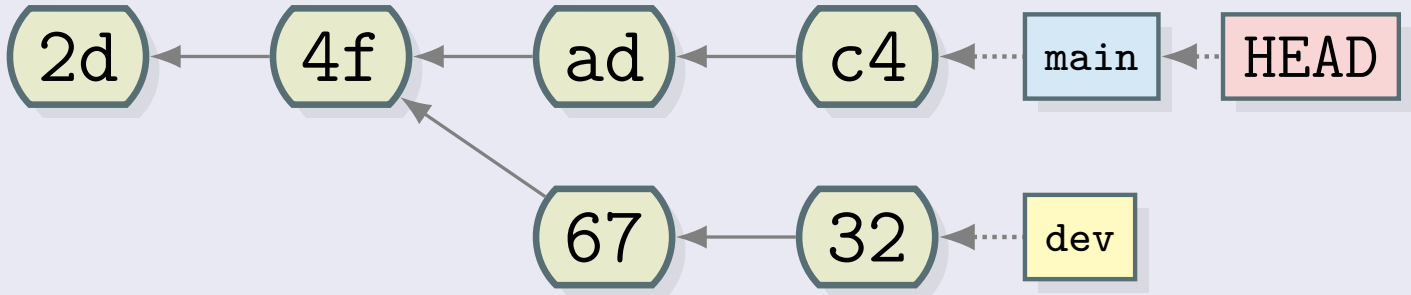
Exercice V - Préparation fusion et rebase

1. Ajouter un nouveau commit dans la branche **dev**

Préparation fusion et rebase

```
1 $ git switch dev            # Goto dev branch
2 $ vim README.md             # Modify README
3 $ git commit -am "devC1" # Commit 1 in dev branch
```

État actuel du dépôt local



Comment récupérer les modifications de **dev** dans **main** ?

- Les branches **dev** et **main** ont divergées
- Problème de fusion de branches : gestion des conflits

Exercice VI - Fusion de branches avec conflits

1. Fusionner la branche dev dans la branche main en local

Fusionner : la commande merge

```
1 $ git switch main   # Goto main branch
2 $ git merge dev     # Merge dev branch to main branch
```

```
~/Documents/GIT/FirstRepository on main merge ~1
git status
On branch main
Your branch is up to date with 'origin/main'.

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

~/Documents/GIT/FirstRepository on main merge ~1
cat README.md
# FirstRepository

First commit from main branch

<<<<<<< HEAD
seconde commit from main branch

third commit from main branch - local
=======
First commit from dev branch
>>>>>>> dev

~/Documents/GIT/FirstRepository on main merge ~1
```

Figure: Résultat de la fusion

Exercice VI - Fusion de branches avec conflits

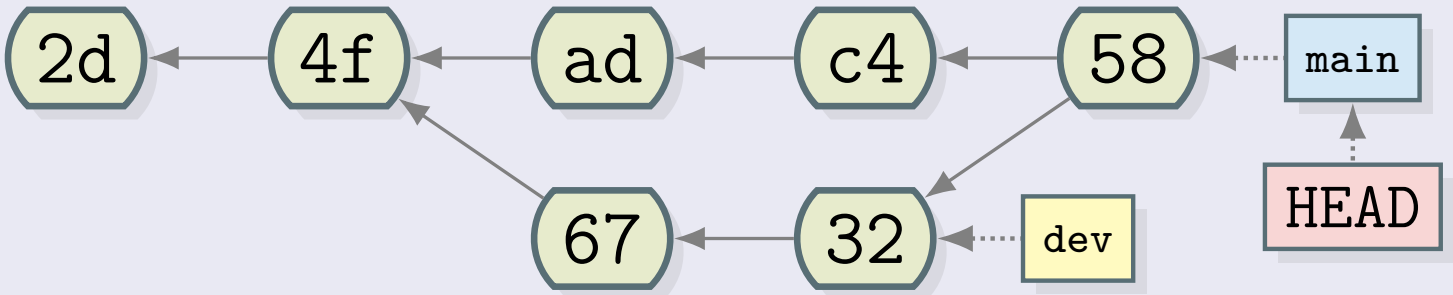
1. Fusionner la branche dev dans la branche main en local
2. Modifier le ficher README.md
3. Ajouter ce fichier au staging (marquer la résolution de conflits)
4. Faire le commit

Fusionner : la commande merge

```
1 $ git switch main     # Goto main branch
2 $ git merge dev       # Merge dev branch to main branch
3 $ vim README.md       # Modify README.md
4 $ git add README.md   # To mark resolution
5 $ git commit -m "dev->main - Conflicts resolution"
```

Ne pas hésiter à utiliser la commande git status régulièrement

État du dépôt local après la fusion - Merge



Les commits de la branche *dev* ont été fusionnés avec la branche *main* dans le dernier commit de cette branche (58)

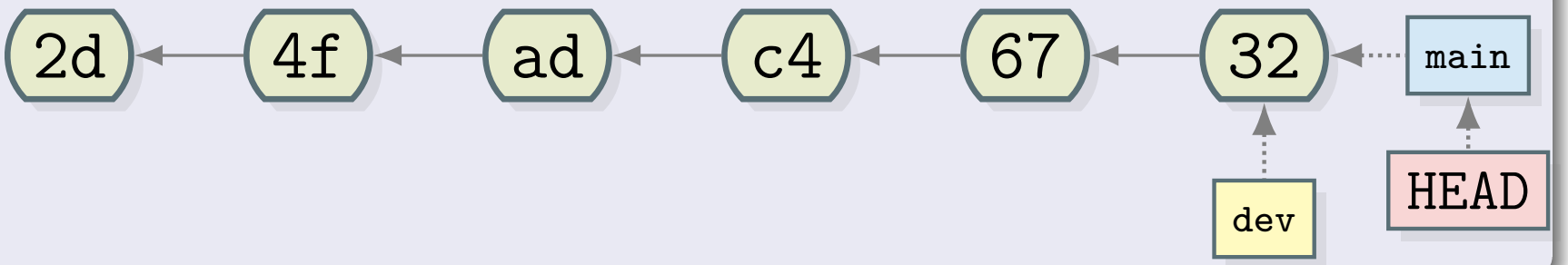
Exercice VII - Rebase de branches avec conflits

1. Revenir dans l'état du dépôt distant (supprimer puis cloner de nouveau le dépôt distant)
2. Ajouter un nouveau commit dans la branche **dev** en local
3. Rebaser la branche dev dans la branche main en local
4. Modifier le ficher README.md
5. Ajouter ce fichier au staging pour marquer la résolution de conflits
6. Terminer le rebase

Rebaser : la commande rebase

```
1  $ git switch dev            # Goto dev branch
2  $ vim README.md             # Edit readme
3  $ git commit -am "devC1"    # New commit in dev
4  $ git switch main           # Goto main
5  $ git rebase main dev       # Rebase dev in main
6  $ vim README.md             # Resolve conflicts
7  $ git add README.md         # Mark resolved
8  $ git rebase --continue     # Continue rebase
9  $ git switch main           # Goto main
10 $ git merge dev             # Merge dev in main
11 $ git branch -d dev         # Delete dev (-D?)
```

État du dépôt local après le rebase



Tous les commits de la branche *dev* ont été mis à la suite de ceux de la branche *main*

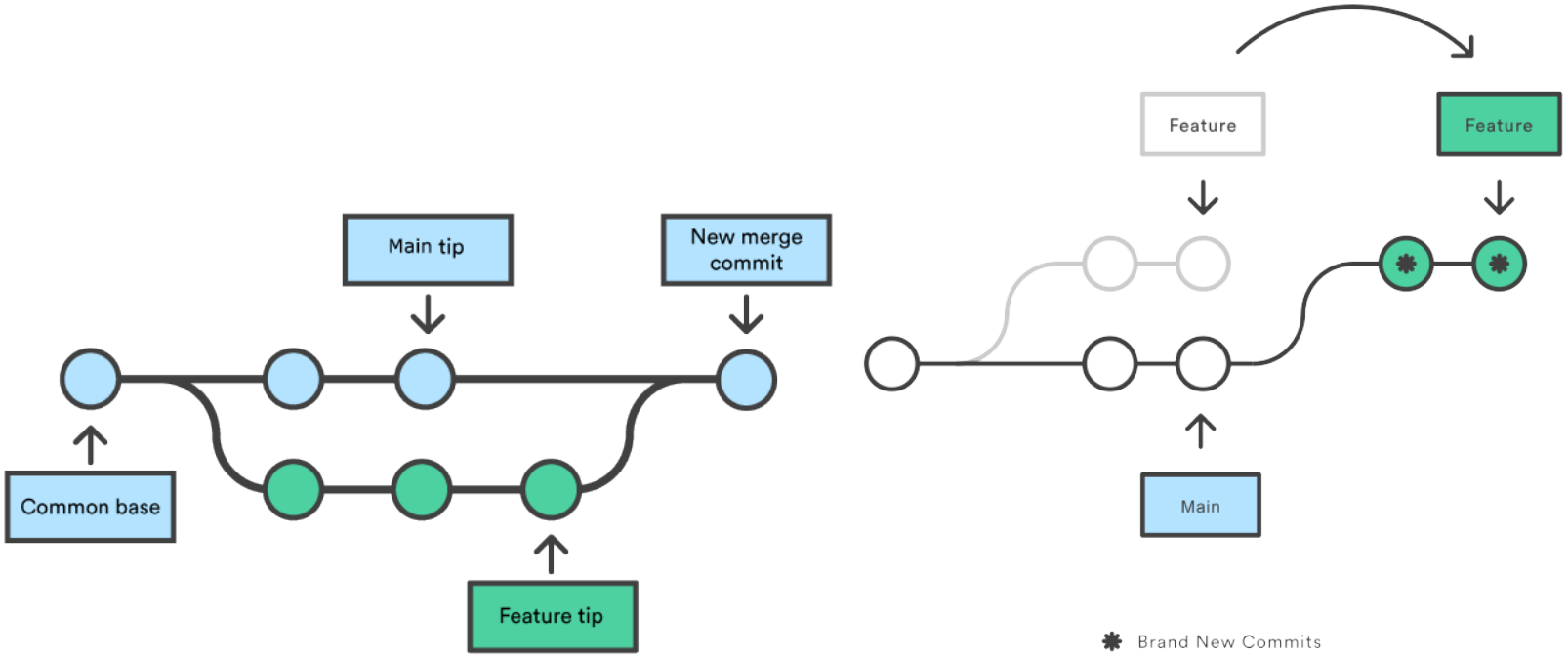


Figure: Illustration des commandes merge à gauche et rebase à droite

Petit résumé intermédiaire

Fusionner : la commande merge

```
# Merge branchName in current branch
$ git merge branchName
```

Rebaser : la commande rebase

```
$ git rebase main feature # Rebase feature in main
$ ..... # Resolve conflicts ?
$ git rebase --continue # Continue rebase
$ git switch main # Return to main
$ git merge feature # Merge feature in main
$ git branch -D feature # Delete feature
```

Travail en équipe

- Chaque développeur doit avoir un compte GitHub
- Chaque développeur doit avoir un PAT
- Le propriétaire du dépôt origine doit envoyer des invitations

Pour plus d'infos : **Inviter des collaborateurs**

2 Annexes

- Console, Terminal : Command Line Interface (CLI)
- Oh my ¿what? !
- Applications utiles
- Créer un dépot Github à partir d'un dépot local
- Lecture de la sortie d'un diff
- Lecture de la sortie d'un pull
- Une longue ligne de commande !
- Les accroches - Hooks
- Les alias
- Cueillons des cerises !
- Pour finir !

Console, Terminal (Win, Linux, OSX) - Commandes de base 1/2

```
1  $ pwd                    # Print working directory
2  $ cd                     # Idem (Win)
3  $ mkdir folder           # create folder directory
4  $ rm folder              # Remove folder directory
5  $ rm -rf folder          # Idem - Recursive, force
6  $ rmdir folder           # Remove folder directory (Win)
7  $ rmdir /S /Q folder     # Idem - Recursive, force (Win)
8  $ cd Desktop        # Change current directory to Desktop
9  $ cd ..             # Change current directory to parent
10 $ cat fileName      # Print fileName content to Terminal
11 $ more fileName     # Idem
12 $ type fileName     # Idem (Win)
```

Console, Terminal (Win, Linux, OSX) - Commandes de base 2/2

```
# Set "Some text" in fileName (Delete, Create before)
$ echo Some text > fileName
# Append "Some text" in fileName (Create before)
$ echo Some text >> fileName
$ vi fileName          # Edit fileName
$ notepad fileName     # Idem (Win)
$ ls                   # list current directory content
$ dir                  # Idem (Win)
$ ls -la               # Full info and all
```

Pour avoir un superbe terminal !

Suivre ce lien : 👍 **Oh my posh !** 👍

Installation sous Windows

Pour avoir un autre superbe terminal !

Suivre ce lien : **Oh my bash !**

Installation :

```
$ bash -c "$(curl -fsSL https://raw.githubusercontent.
    com/ohmybash/oh-my-bash/master/tools/install.sh)"
```

Désinstallation :

```
$ uninstall_oh_my_bash
```

Modification du thème et des plugins du Terminal alors possible

Encore une alternative : **Oh my zsh !**

Figure: Logiciels SourceTree, GitHub et GitKraken Desktop

Trois outils graphiques libres et très utiles

Contexte

J'ai des fichiers sur mon ordi que je souhaite gérer sur GitHub

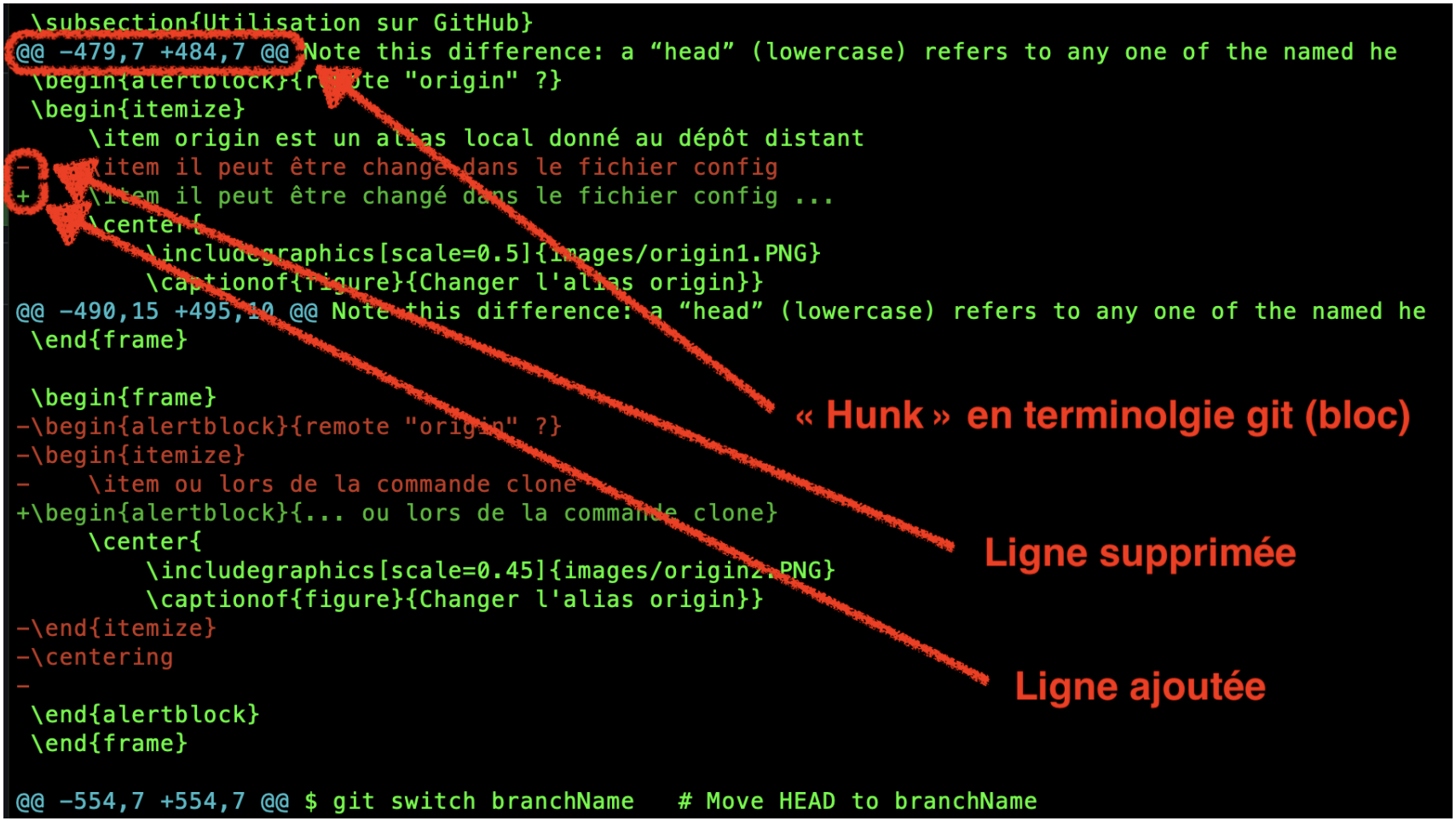
Solution

1. Créer le dépôt local si pas déjà fait
2. Créer le dépôt distant sur GitHub **VIDE** (évite les erreurs) :
   - Pas de fichier README.md, .gitignore, licence etc.
   - Récupérer l'URL du dépôt distant
3. Ajouter cet URL en tant que dépôt distant
4. Pousser la branche principale sur le dépôt distant

```
1 $ git init
2 $ git add ...
3 $ git commit ...
4 $ git remote add origin REMOTE-URL
5 $ git remote -v # Check URL has been added
6 $ git push -u origin main
```

Lecture de la sortie d'un diff

```
$ git diff 7e46066..3a78b4a # Changes between commits
```



-479,7 : 7 lignes **à partir de** la 479 dans le commit 7e46066

+484,7 : 7 lignes **à partir de** la 484 dans le commit 3a78b4a

```
1 # Changes between workspace and stage
2 $ git diff
3 # Changes between workspace + stage and last commit
4 $ git diff HEAD
```

```
diff --git a/presentation.pdf b/presentation.pdf
index eb2bfe9..326de97 100644
Binary files a/presentation.pdf and b/presentation.pdf differ
diff --git a/presentation.tex b/presentation.tex
index a02e9fe..4f1635d 100644
--- a/presentation.tex
+++ b/presentation.tex
@@ -1165,36 +1165,6 @@ Pour plus d'infos : \textcolor{blue}{\textbf{\unc
 \end{frame}

 \section{Les projets}
-
-
-\begin{frame}
-\begin{alertblock}{Rien à disposition pour faire des mesures objectives
-\begin{itemize}
```

Les index présents en entête de la sortie d'un diff sont des identifiants internes à git, pas des numéros de commits !

Comparaison de fichiers entre dépôts distant et local

```
# - Workspace and stage diffs are ignored -
# Without git fetch, changes between origin and local
# repos won't be synchronised and diff will be blind.
# Use git branch -a to list local and "remote local"
# branches. e.g. : branchName and
# remotes/origin/branchName
# In order to update "remote local" branches ->
$ git fetch
# Changes between main branches : local branch main
# and "remote local" branch remotes/origin/main ->
$ git diff main origin/main
# Changes in path/tofile between origin
# and local repos in main branches ->
$ git diff origin/main:path/tofile main:path/tofile
$ ...
$ git merge origin/main
```

Lecture de la sortie d'un pull

```
~/Documents/GIT/CourseMaterial  on main                                              ✓  base
└ git pull
From https://github.com/MasterFHGit/CourseMaterial
   7e46066..3a78b4a  main       -> origin/main
Updating 7e46066..3a78b4a
Fast-forward
 images/GUI.png              | Bin 0 -> 24545 bytes
 images/SaveCentralNew.png | Bin 0 -> 65138 bytes
 images/SaveDistribNew.png | Bin 0 -> 111885 bytes
 images/SaveLocalNew.png   | Bin 0 -> 26221 bytes
 images/devis.png          | Bin 0 -> 75387 bytes
 images/zones1.png         | Bin 0 -> 54719 bytes
 images/zones2.png         | Bin 0 -> 60191 bytes
 presentation.pdf          | Bin 5267356 -> 6625085 bytes
 presentation.tex          | 324 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++---------------
 presentation.vrb          |  19 ++++-
 10 files changed, 259 insertions(+), 84 deletions(-)
 create mode 100644 images/GUI.png
 create mode 100644 images/SaveCentralNew.png
 create mode 100644 images/SaveDistribNew.png
 create mode 100644 images/SaveLocalNew.png
 create mode 100644 images/devis.png
 create mode 100644 images/zones1.png
 create mode 100644 images/zones2.png
```

De nouveaux fichiers ajoutés en mode normal (100644)

343 changements (ajouts et suppressions)

Beaucoup de changements dans presentation.tex

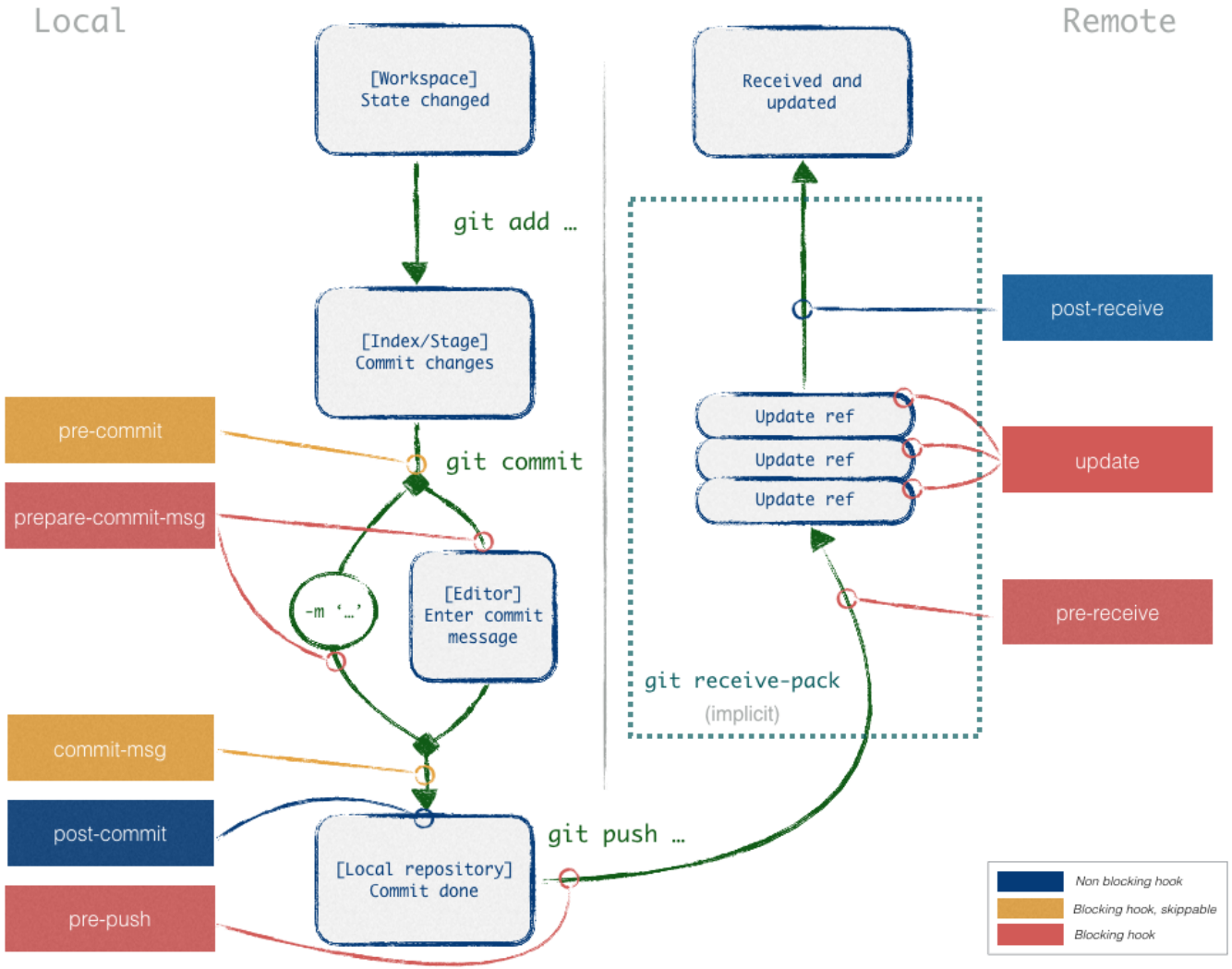
Il existe d'autres modes : e.g. 100755 et 120000

Une longue ligne de commande !

```
* 2186499 Update reset and go further topics
*   af46e63 Merge conflict
|\
| * 7e46066 Add to go further
| * 654b4e9 Change font size and add git SCM link
* | fd02601 Add clone command with uname and passwd
|/
* db8324e Remove pong image
* 7149293 Update revert and reset commands
* e8896ef Update merge and rebase commands
* 48b507e Global update 3
* 6b16542 Global update 2
* e44f652 Global update
* fe4cd53 Ajout des commandes rebase et merge
* b6f584b Add annexes
*   74a5511 Resolve conflict
|\
| * 8971e4d Some minor changes
* | e24286b Update projects
|/
* de0dca4 Remove spaces before $ in lstlisting env.
* dedb9fb Add credential and Signed-Off alerts blocks
* 8ff1a7f Update projects part
* 73cc805 Update projects and git commands
* f6ae20c Add main concepts slides
* c979f94 Add gitDAG library to draw nice git graph
```

```
1 $git log --graph --decorate --abbrev-commit --all --pretty=oneline
```

Un superbe graphe de commits dans la console ☺!



Documentation Hooks sur Delicious Insights

Documentation Hooks sur Git

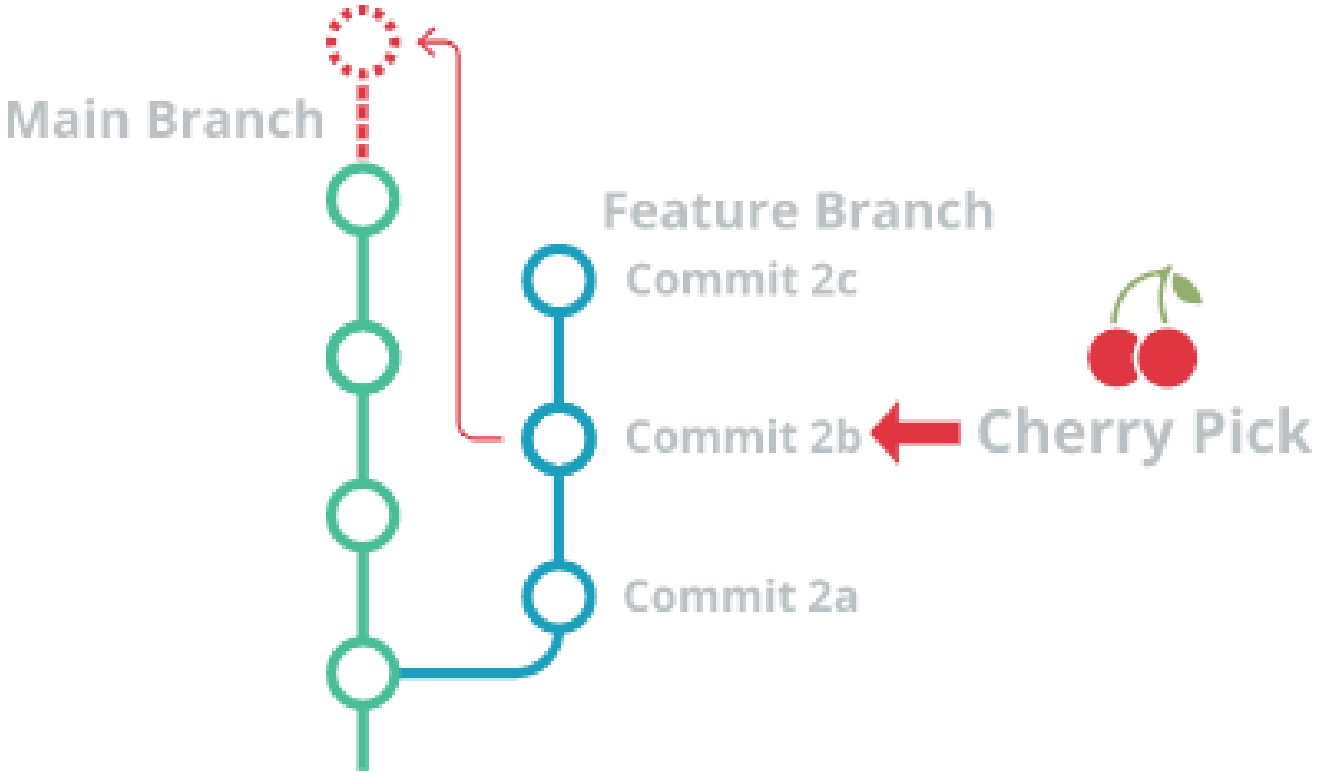
Git Alias

```
alias gs='git status'
alias ga='git add'
alias ga='git push'
alias  c='git commit'
```

Documentation Alias sur Git

Les alias en pratique

```
1 # Creating an alias to get a
2 # graph of commits in console
3 $ git config --global alias.lg 'log --graph --decorate
    --abbrev-commit --all --pretty=oneline'
4 $ git lg
```



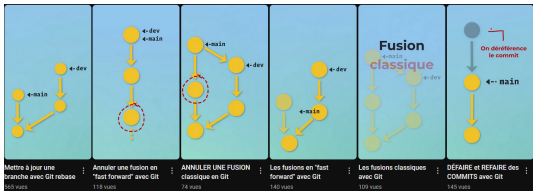
"Cherry-pick" sur Delicious Insights

Cherry-pick en pratique

Une bonne pratique consiste à créer une branche de report. La Gestion de conflits/fusions est possibles. L'option -x permet d'écrire l'ID du commit initial dans le commit final ("cherry picked from commit commitID").

```
$ git switch rBranch # Branch which will receive picks
# Create and switch to a report branch from rBranch
$ git switch -c rBranchReport
$ git cherry-pick -x commitID1 # Pick oldest commit
$ git cherry-pick -x commitID2 # Pick next commit
$ git cherry-pick -x commitID3 # Pick next commit
$ # etc ... Manage possible conficts etc...
$ git switch rBranch # Switch to rBranch
$ git merge rBranchReport # Report rBranchReport
$ git branch -d rBranchReport # Delete rBranchReport
```

Pour finir !



Shorts Git (sur Delicious Insights)

les formes de HEAD expliquées sur StackOverflow

"Ignorer des fichiers" (sur Delicious Insights)

"La remise" (sur Delicious Insights)